SIGS

Get Ready for **XML One** p. 63 • MS Vies for Embedded Space p.12

Java 2
Core Class Libraries
FREE POSTER
SPONSORED BY SYMANTEC
see page 59

Java™ REPORT

April 1999
Volume 4, Number 4
$4.95 US/$7 Can/$4.95 Int'l

THE INDEPENDENT SOURCE FOR JAVA DEVELOPMENT • www.javareport.com

Spotlighting the New APIs

- Using the Collection API as the Foundation of Your Data Structures
- Directly Interact with Applications Through JavaHelp
- Why Immutable Objects are Crucial For Designing Durable APIs

How to Use Exception Handling Effectively

Database Selection Strategies For Long-Term Success

Creating JavaScript Objects For Your Web Pages

Two Approaches to Solving Versioning Problems

0 74470 88941 0 04

REVIEWED
ILOG's JViews

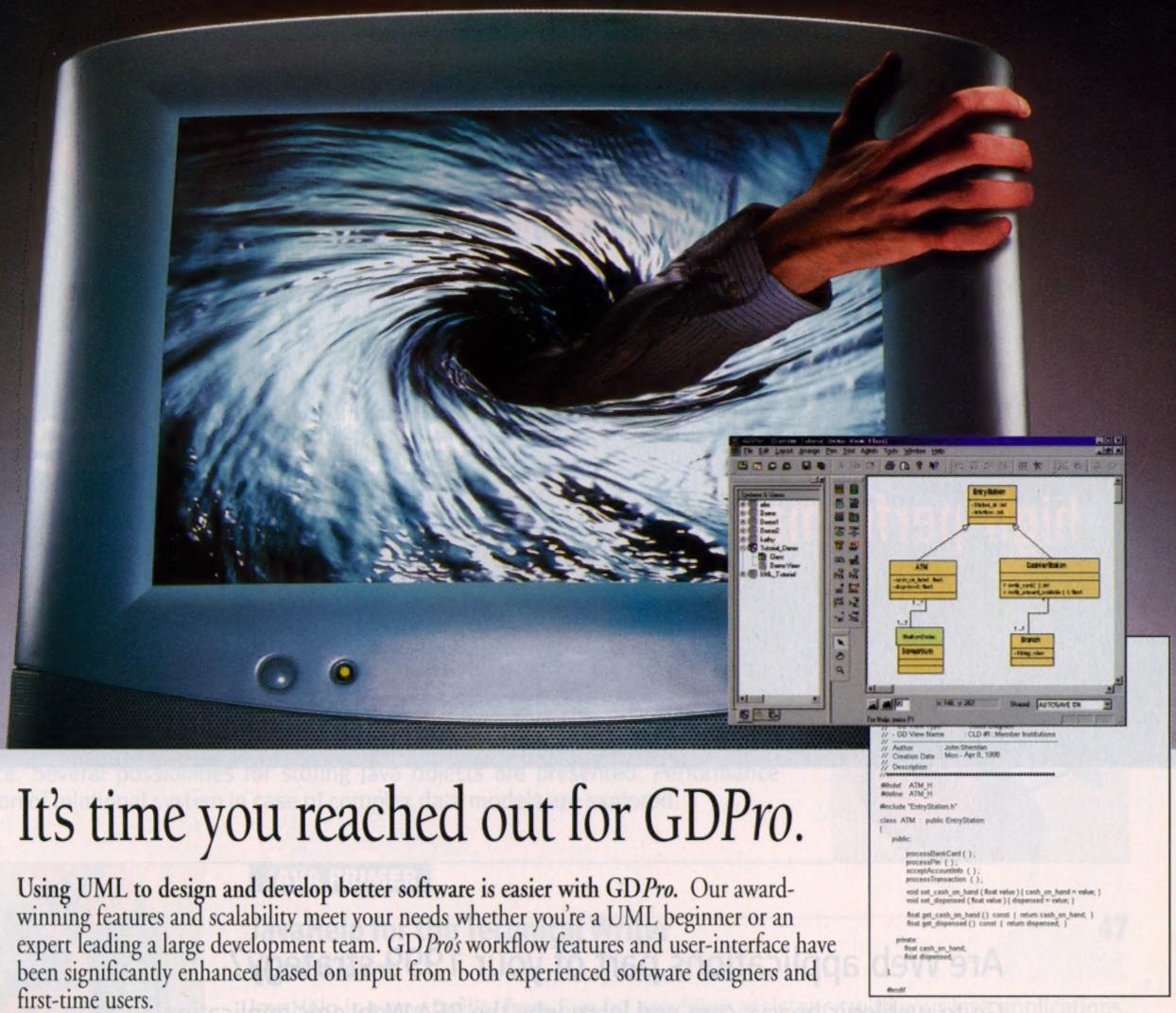
4thpass

SourceGuard3.0

Smart Bytecode Protection

[www.4thpass.com]

Are you asked to make your application faster, better, Internet-enabled, reusable, distributed, customer-oriented, component-based, "reengineered", *and* well-documented?



It's time you reached out for GD*Pro*.

Using UML to design and develop better software is easier with GD*Pro*. Our award-winning features and scalability meet your needs whether you're a UML beginner or an expert leading a large development team. GD*Pro's* workflow features and user-interface have been significantly enhanced based on input from both experienced software designers and first-time users.

The GD*Pro* tool kit includes all the features you need, including some that the competition simply can't match. Robust reverse-engineering and round-trip engineering are achieved without the use of cumbersome code markers. Enterprise developers love the collaborative "whiteboard" design facility and true team-based development support.

With GD*Pro*, you get a visual tool kit which works the way you do, not one which forces you to work around legacy tool technology. Plus you are not forced into a proprietary tool suite - GD*Pro* integrates with most other popular tools which you already use (like requirements management), or which you plan to use in your development environment.

When it comes to Software Analysis, Design, and Reengineering, you've got a lot at stake, and not all tools are what they appear to be. So you owe it to yourself...

Reach out for GD*Pro* 3.0 for 30 days...at no charge!

Call 800-811-ASTI (2784) today, or visit our web site at: www.advancedsw.com, or email us at: info@advancedsw.com

ADVANCED
SOFTWARE TECHNOLOGIES INC.

"Best Tools. Best Service. No Compromise."

Are Web applications part of your 1999 strategy?

Go to weblogic.beasys.com and learn why the BEA WebLogic application server has been touted by the industry as the leading solution for quickly deploying and managing your high-performance, mission-critical, Java-based, enterprise-scale (etc.) Web applications.

Crossroads A-List Award:
"Rapid Application Development Tools for Java" (January 1999)
JavaWorld Editor's Choice Award:
"Best Web Application Server" (December 1998)
Java Developer's Journal Editor's Choice Award:
"Best Web Application Server" (December 1998)
Intelligent Enterprise RealWare Award:
"Best Application Using a Component Architecture" (December 1998)

BEA WebLogic®

weblogic.beasys.com

© 1999 BEA Systems, Inc. All rights reserved. BEA, the BEA logo, and WebLogic are registered trademarks of BEA Systems, Inc. Java is a trademark or registered trademark of Sun Microsystems, Inc. Other trademarks are the property of their respective owners.

BEA

April 1999
Vol. 4, No. 4

Java™ REPORT

THE INDEPENDENT SOURCE FOR JAVA DEVELOPMENT • www.javareport.com

Technical Features

JAVA PRIMER

The Collection API 17

John Hunt

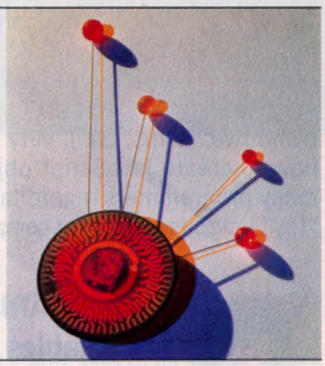
The Collection API is implemented as part of the java.util package. The new collection classes extended the facilities provided by the utility classes Vector and Hashtable. For example, ArrayList provides functionality similar to existing data structure classes such as the Vector class, but is optimized for performance. Like Vector and Hashtable, the collection classes can only hold objects, so, if you wish to hold the basic types within them, you need to wrap the basic type in the appropriate object wrapper.

ENTERPRISE JAVA

How to Store Java Objects? 33

Tomaz Domajnko, Matjaz B. Juric, Bostjan Brumen, and Ivan Rozman

Advantages accruing from the development process compared to conventional programming languages are well known—Java is platform independent allowing simple multiplatform system development with the opportunity to provide efficient and powerful data persistence. Several possibilities for storing Java objects are presented. Performance degradation of relational system in case of complex data models are explored.



JAVA PRIMER

JavaHelp for the Technical Writer 47

Mark Fisher

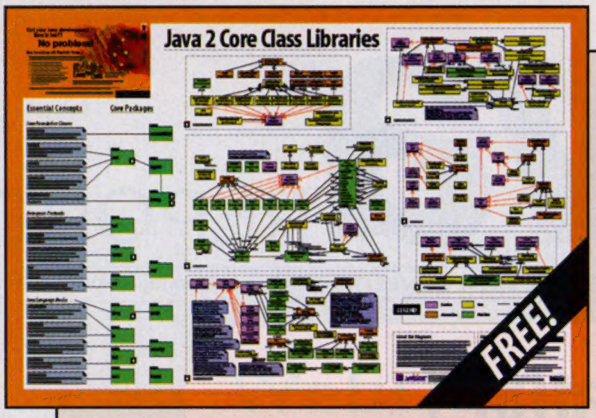
JavaHelp is a new facility from Sun for providing assistance with your Java applications. JavaHelp provides a browser-like help viewer but without the browser. Because no browser is used, there are no browser dependencies. There is no need to consider the widely differing capabilities of the browsers that customers might be using or worry about individual browser quirks. It is written entirely in Java to provide consistent help across all platforms for full control of the help environment.

POWER JAVA

Effective Exception Handling in Java 55

Peter Haggar

Exception handling is a powerful and useful feature of the Java language. With this power comes a degree of complexity that must be understood to effectively utilize exception handling in Java programs. The language introduces new concepts to the exception handling model that make exceptions both easier and more difficult to use properly.



Our latest illustrated version detailing the core concepts of the Java 2 platform. For more information on how to receive your free copy, visit www.javareport.com/poster.

Java™ REPORT
THE INDEPENDENT SOURCE FOR JAVA DEVELOPMENT • www.javareport.com
April 1999 Vol. 4, No. 4

Departments

Editor's Note 6
Will the increase ever stop?
Dwight Deugo

Letters to the Editor 8

Java Buzz 10
Philip J. Gill
Sun Lets Jini Out of the Bottle...and Microsoft Strikes Back; Sun Beats Back Rival Embedded Java Standard; Flashline.com Offers First Online Auction of Developer Services; Sun Gains HAVi Backers in Java Battle; W3C Releases New Java Web Server; and more.

Product Review 14
Class Libraries That Integrate Into the Existing AWT
JViews encourages 2D developers to take Java plunge
Todd C. Zino/NSTL
Java's advantages in designing applet user interfaces earmarked it increasingly for use in two-dimensional data plotting. ILOG has developed JViews as a port of its long-standing C++ Views package. The idea is to provide functionality for handling map and graph data, prototyping arrangements of graphics in precise form and binding to underlying data. JViews is in a leading position to encourage high-end 2D developers (who've no doubt been eyeing Java curiously for a long time) to take the plunge!

Ad Index 56

Columns

JavaScripting 66
Your own private Idaho()
Steven W. Disbrow
One of JavaScript's greatest strengths and most overlooked features is its object-oriented nature. While almost all JavaScript programmers make use of JavaScript's built-in objects (windows, forms, etc.), very few take the time to learn how to create JavaScript objects of their own. In fact, many programmers regard JavaScript as something of a "toy" language, unfit for serious programming.

Durable APIs 70
Immutables
Mark Davis
Designing for larger-scale systems can be quite different than designing for small programs. The cost of making incompatible changes becomes very high. You have to design your code to be durable—where the APIs can withstand the test of time, and can grow when you need to extend them, without breaking your clients.

Scott's Solutions 80
Versioning solutions
Scott Oaks
Two approaches that can help solve the versioning problem are examined. Versioning issues creep into several areas of Java. When objects are deserialized, there is a potential versioning problem if the implementation of the serialized class has changed; there are version numbers in the Java class file specification that can come into play when you deal with low-level issues of the virtual machine, and there are probably others as well.

Check out Java Report Online!

- *Selected features and columns from the print edition*
- *All code listings are available online!*
- *Monthly product updates*
- *Coming soon: Java Report Archives*

Cover image by Roy Wiemann/Image Bank

e-business

Get a **free Web Self-Service Starter Pack,** including product trial code, and find out more about creating hot self-service apps quickly and easily, at www.software.ibm.com/websoftware

And for your next trick, make this line disappear.

Customers using the Web to get what they need for themselves. That's the idea behind Web Self-Service. But first, you've got to grab them with hot self-service apps. Enter Web Self-Service solutions from IBM, with great products like WebSphere™ Studio software. Because it's Java™-based, its servlet creation wizards eliminate tedious CGI script and let you quickly build dynamic, interactive Web apps. And you can create scripts for Java, JavaScript, Jscript, HTML, D-HTML and JavaServer pages with NetObjects ScriptBuilder tools. All while NetObjects Fusion lets you visually create, edit and manage your entire Web site. The result: you can instantly preview your work, smoothly add language elements and quickly navigate to embedded functions and objects. Once you've created your apps, WebSphere Application Server lets you host Java servlets on most Web servers, with built-in connectors to tap data and apps you're already using. And if you need to build collaborative workflow applications for your rapidly changing business processes, Lotus® Domino™ Application Server supports secure, highly interactive Web Self-Service solutions. So creating hot Web apps has never been faster or easier. Find out more about these superior products at www.software.ibm.com/websoftware

IBM

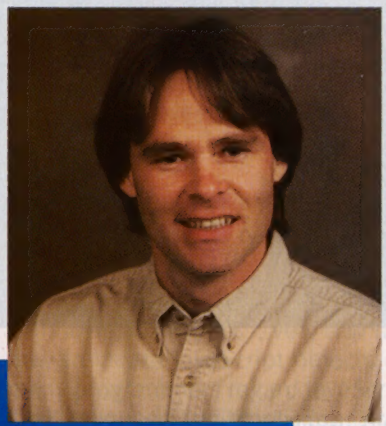
Solutions for a small planet™

IBM, WebSphere, the e-business logo and Solutions for a small planet are trademarks of International Business Machines Corporation in the United States and/or other countries. Lotus and Domino are trademarks of Lotus Development Corporation in the United States and/or other countries. Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States and/or other countries. Other company, product and service names may be the trademarks or service marks of others. © 1998 IBM Corporation. All rights reserved.

Will the increase ever stop?

IT'S BEEN AN interesting couple of months for email. The number I receive daily has increased by more than 50% since early February. I can attribute some of this increase to the following reasons.

First, people have become so passionate about Java that they defend it without reservation. For example, Tig Tillinghast's column in the January issue ("How the Web is Killing Java") seemed to touch a nerve with a small percentage of readers, and they were sure to let me know about it. On the other hand, there were those who were not supportive. Take for example, the case where in one article someone found it surprising that I would let anyone write that "Java is a fully OO language." You and I know we have to deal with those "nasty" primitive types when building our applications, but everything else sure looks like objects to me.



Dwight Deugo
deugo@scs.carleton.ca

Mixed between these extremes were the emails I received about people wanting to get involved with the Java 2 Platform. People were asking about it, willing to write about it, wishing to describe the products they are building with it, or just hoping to promote products that help them use it.

Love it, hate it, or just use it to pay the bills, it is hard to ignore the fact that Java and object-oriented programming help us to manage the complexity of our applications. How you ask? They provide us with the ability to build abstractions. Like me, I am sure you are glad you don't often have to use assembler anymore. However, you should make sure you understand the reason why. When I programmed in Pascal or C, I always focused on building good abstractions with both data and procedures. Now, I easily combine the two and call them an object. Do your objects capture good abstractions? How many abstractions should a single object represent? 0, 1, 2, more? One is the right answer to me.

Remember, object-oriented systems are built with many collaborating and communicating objects, not with a single class containing a static main method.

Even if you get your abstractions right, there are other entities lurking inside them. These entities are the computational processes. I am not talking about processes in terms of threads, I am talking about how your applications and objects evolve and manipulate other abstractions. For example, do you know if Java supports tail-recursion? If you don't, you better be careful using recursion on those large graph objects. Lucky for us, the Java 2 Platform has an enormous set of abstractions, and I am sure Sun has done a great job keeping those computational spirits under control. I just hope I can find the right abstraction when I need one. I do hate writing my own when there are so many to choose from.

Dwight Deugo

Java™ REPORT
THE INDEPENDENT SOURCE FOR JAVA DEVELOPMENT

BPA INTERNATIONAL

Editor-in-Chief
Dwight Deugo; *deugo@scs.carleton.ca*

Senior Managing Editor
Kathleen M. Major; *kmajor@sigs.com*

Senior Art Director
Andrea Cammarata; *acammarata@sigs.com*

Assistant Managing Editor/ Associate Online Editor
Dan Olawski; *dolawski@sigs.com*

Assistant Art Director
Fernando Quiñones; *fquinones@sigs.com*

Editorial Assistant
Melissa Shepherd; *mshepherd@sigs.com*

Associate Editor
Philip J. Gill; *philipgill@aol.com*

Contributing Editor
John K. Waters; *john@watersworks.com*

Product News
Jane Grau; *jgrau@hudsonet.com*

Editorial Intern
Douglas J. Lee; *dlee@sigs.com*

COLUMNISTS
Steve Ball, *Independent Consultant*
Mark Davis, *IBM*
Steven Disbrow, *EGO Systems*
David S. Frankel, *Genesis Development Corp.*
Vincent Hardy, *Sun Microsystems*
Lowell Kaplan, *Random Walk Computing*
Craig Larman, *Object Space*
John Miller Crawford, *Dave Clark Design Associates*
Scott Oaks, *Sun Microsystems*
Tig Tillinghast, *Internet Profiles*

SIGS PUBLICATIONS, INC.
Richard P. Friedman, *Founder, President, & CEO*
Margherita R. Monck, *Vice President Finance/Circulation*
Gayle C. Ehrenman, *Executive Editor*

SIGS ADVISORY BOARD
François Bancilhon, *Ardent Technology*
Grady Booch, *Rational Software Corp.*
Alan Brown, *Sterling Software*
Stuart Frost, *SELECT Software Tools*
Thomas Keffer, *Rogue Wave Software*
Bertrand Meyer, *ISE*
Meilir Page-Jones, *Wayland Systems*
Jan Popkin, *Popkin Software and Systems, Inc.*
Kenneth S. Rubin, *Innolution*
Dave Thomas, *Object Technology International*

SIGS PUBLICATIONS

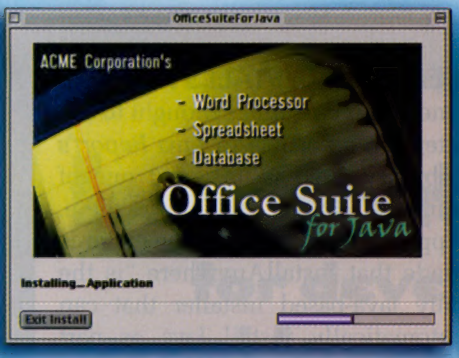
Publishers of *JOOP, Component Strategies, C++ Report, Java™ Report, OBJEKTSpektrum (Germany), Java™SPEKTRUM (Germany), Application Development Advisor (UK), Java™ Report Online, Component Strategies Online, C++ Report Online, and JOOP Online.*

Zero G Software's InstallAnywhere™ 2.5

New Features Include:

- SpeedFolders™ - faster-than-native installations
- Java 2 support
- Optimized for large installations
- Install with or without a Java™ VM
- LaunchAnywhere™ applications launchers for: Windows 95, 98, NT, Mac OS, OS/2, Solaris, Linux, HP-UX, IRIX and AIX

Professional Look and Feel



Installers have a professional look and feel that's familiar to all users on all platforms. You can run installers from CD-ROM or directly from the Web.

JDJ Java Developer's Journal JDJ Editor's Choice Award

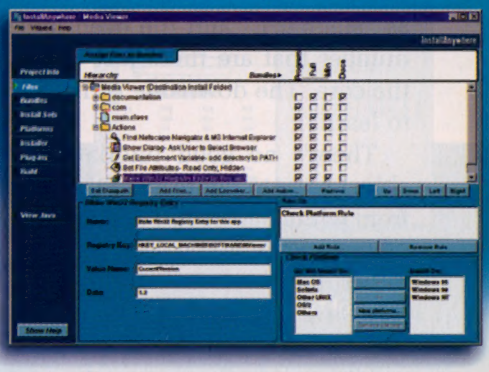
Editors' Choice JavaWorld 1998 WINNER

Create Installers Quickly

InstallAnywhere Project Wizard - OfficeForJava
set classpath
Select items and click "In Classpath" to add them to the classpath. Alternatively, press the "Auto Set Classpath" button to have the Wizard set the classpath for you.
Set Classpath
OfficeSuiteForJava (Destination Install Folder)
help
OfficeSuite.zip
In Classpath
Indicates item is on the classpath.
Automatically Set Classpath
Exit
Advanced Mode...
< Back
Next >

In six steps you can create an installer with the Project Wizard. It sets the classpath and finds "main" classes automatically.

Power and Control



Need more power and control? The Advanced Designer lets you control everything! Create custom install sets, icons, installation rules, multiple destinations and more...

InstallAnywhere Standard Edition $995
InstallAnywhere Express Edition $495

New Optional Add-ons:

- Internationalization East
- Internationalization West
- Advanced Actions Pack

zero SOFTWARE

FREE TRIAL VERSION available at www.ZeroG.com

Zero G Software 118 King Street, Suite 415 San Francisco, CA 94107-1916 Phone: (415) 512-7771 Fax: (415) 512-7775

WANTED: MORE TUTORIALS!

I was a long-time reader of *C++ Report* before I began working primarily with Java. I hope to be a long-time reader of *Java Report*, and have been reading it since the first issue. However, it has not always given me what I want, as the *C++ Report* always did—so I thought I'd tell you what would be useful to me.

I am a senior software developer. My job is to design software, figure out what tools and packages to use, and code the more difficult or critical components. I also assist junior developers.

Despite my level of skill, I find that I spend an inordinate amount of time writing "hello world" programs.

This is the problem I face: I spend a great deal of time just figuring out what the available tools and packages are, learning the vocabulary by writing one "hello world" program after another.

Java's most important contribution to the software world is the phenomenal number of class libraries and tools it has provided as "standard," and the even larger number that are there just outside the core. The downside is: it's a lot to learn.

There's no good tutorial available. I have to figure everything out from white papers full of jargon and materials intended for reference rather than tutorial reading (e.g., *JavaDoc*). If I'm lucky, I'll get some well-commented sample code.

I think *Java Report* could be part of the solution. I doubt I am alone; many Java programmers do not use the tools and packages applicable to their work because they do not know they exist, do not understand them, and think they do not have time to learn.

I think *Java Report* should concentrate on providing tutorial information about common, widely available tools and packages.

You should target highly skilled developers even if you're covering what seems like a basic API. With the number of packages and tools entering the Java landscape, we're all starting from scratch, all the time.

In a tutorial I would expect to learn right away what problem the package was intended to solve. Next, I'd want to see how "hello world" works, followed by a discussion of the traps and pitfalls I'll face as I go forward. I also want to know how it relates to other packages, coupled with an exploration of the more advanced things the technology can do. In the end, I want to know what kind of performance I can expect and what I have to do to get it to scale up for heavy use.

Justin Wells
justin@fever.semiotek.com

Editor's Reply:
We always like to hear readers' views on the magazine and its direction. Throughout 1999, we plan to do much of what you indicated. We will focus on describing and evaluating different tools, products, and technology, and you will see more "lessons" on how to use the various packages in Java: Servlets, Collections, Mail, and Help, to name but a few, have already scheduled. You will also see a new column soon on Microsoft's view of Java.

Java Report *strives to be your independent source of information on Java. We do not choose sides—we report on them. Keep telling us how we are doing. We want to continue to be the best magazine on Java technology.*

MULTIPLICATION FACTORS

I am new to Java and currently receive *Java Report*. I am trying to write a program that will print out all the multiplication factors of a given integer. An example would be if I ran java myprogram 12 at the command line and hit <enter>, the result would print out 2 2 3 (15 would print out 3 5, 30 would print out 2 3 5, 19 wouldn't print out anything, etc.). Is there any code examples that I can browse to get a feel for this?

Joe Costa
Costajl@aol.com

CLIMB EVERY MOUNTAIN....

Hi. I just compile Java code in a boring office with boring people who talk about computers and sleep with computers. But...I just discovered the picture in the February 1999 issue. The picture shows Editor-in-Chief Dwight Deugo rock climbing. Incredible—something human! [I'm a] fanatic climber, well-paid programmer by unlucky circumstance.

Kris Hasenjaeger
Kristian.Hasenjaeger@uibk.ac.at

J'EXPRESS INSTALLS JVM

One of our customers brought to our attention an article in *Java Report*'s February 1999 issue about one of our competitors, InstallAnywhere. Apparently on page 10, the claim is made that InstallAnywhere "is the only Java-based installer that can automatically install Java as part of the installation eliminating the need to download and install Java separately."

Our product, J'Express, also installs the JVM, if needed, before installing a Java product. J'Express has been shipping since 4th quarter 1997. We offer 2 editions: J'Express Standard for $199 and J'Express Professional for $499. Both versions let developers build installers that install the JVM. There are many unique features that are standard in J'Express that aren't offered by the competition or only at an additional price, such as auto-updaters, multilingual installers, and finding classes. A chart comparing J'Express, Install Shield, and InstallAnywhere is at **www.denova.com/JExpress/installer.html**. Thanks.

Marie Hall
DeNova, Inc.
mhall@denova.com

StudioJ

a suite of Java tools

North American pricing
$995
Special upgrade pricing available for existing Rogue Wave and Stingray customers!

- Full source code included!
- No royalties or run-time fees!
- Now with improved JFC support and documentation

Integrated components
for developing great Java™ applications!

Why shop all over for Java components from different vendors and then spend days making them work together? **StudioJ** is your one-stop solution! Now you can get the high-quality, integrated, pure JavaBeans components and classes you need all in one place. To order or upgrade today call 1-800-487-3217 or visit:

www.roguewave.com/ad/studioj

Blend.J for adding over 45 AWT and JFC unique and rich extensions to your GUI front end.

Grid.J for table or spreadsheet layouts, with support for Excel-like formulas, binding to any data source, and much more.

Chart.J for dynamic, customizable charts, with data filters and overlays for composite charts.

DBTools.J for SQL portability across databases, with thread-safe and extensible classes.

Rogue Wave is a registered trademark, and iBabel and .J are trademarks of Rogue Wave Software. Java and all Java-related trademarks are registered trademarks of Sun Microsystems, Inc. All other trademarks are the property of Rogue Wave Software or their respective holders.

Rogue Wave SOFTWARE

Rogue Wave: 800-487-3217 · Stingray: 800-924-4223 · D.A.C.H.: +49-6103-59 34-0 · UK: +44-118-988 0224
France: +33-1-5568 1008 · Italy: +39-02-3809 3288 · Rest of Europe: +31-20-301 26 26

Flashline.com Offers First Online Auction of Developer Services

Flashline.com (www.flashline.com), an online retailer of JavaBeans components, expects to be first with an online auction of Java developer talent. The Cleveland-based company said it will open an online forum at its Web site in which companies can solicit the Java developer community for their services, and receive multiple bids in return. He expects companies looking for Java developers' services to be both corporations looking to developer internal systems, as well as independent software vendors (ISVs) who would resell the software in the open market. It will be called "Beans by Design."

Once a customer posts their proposal, developers will be able to submit bids to win the project. If multiple developers want the job, they could start a bidding contest to see who wins the contract.

Flashline.com president and co-founder **Charles Stack** said the company won't make any money off the online auction forum. Instead, its intention is to stimulate the development of Java components and JavaBeans, some of which may end up later for resale at the company's online retail store.

Four Companies Join Finjan's JSA

Four more companies have joined Finjan's Java Security Alliance (JSA) to help protect companies and individuals from Internet attacks. A security interoperability consortium, the Finjan JSA helps companies explore and solve security issues encountered with Internet mobile code, including Java, ActiveX, or JavaScript. **Finjan Inc. (www.finjan.com)**, based in San Jose, CA, makes a variety of

continued on page 12➤➤

Sun Lets Jini Out of the Bottle

Sun Microsystems, Inc. (www.sun.com) has released its Java-based Jini network connection technology amid a ground swell of support from major industry partners and supporters, ranging from consumer electronics manufacturers to software vendors to service providers.

Jini technology, essentially a lightweight network operating system, provides a simple way for the enterprise and consumer markets to enable instant connectivity of devices to the network. Jini enables any device to participate in a network regardless of the underlying software or hardware. Jini technology sets up basic rules for how to connect to the network, share information, and interact, while remaining transparent to the user. And because Jini technology is platform-independent, devices are no longer limited by specific brands of software, processors, device drivers, or traditional networking protocols.

According to Sun, examples of Jini technology-enabled products and services include office peripherals and consumer connectivity, as well as a host of new opportunities.

In the office, Jini technology-enabled laptops, printers, disk drives, and other devices will be able to connect to ever-changing locations and services in corporate networks, said Sun.

With Jini-enabled consumer connectivity, users can simply connect any device to any network service, such as downloading a movie to a set-top box and using a storage service on the network to store it before viewing at their convenience. Jini should also provide new opportunities for developers and service providers. Using Jini, for example, a media outlet could provide a service that allows for delivery of a customized newspaper to the customer's home printer.

Sun also predicts new business opportunities for device manufacturers: for example, a networked washing machine will be able to detect and solve problems before the user is ever affected.

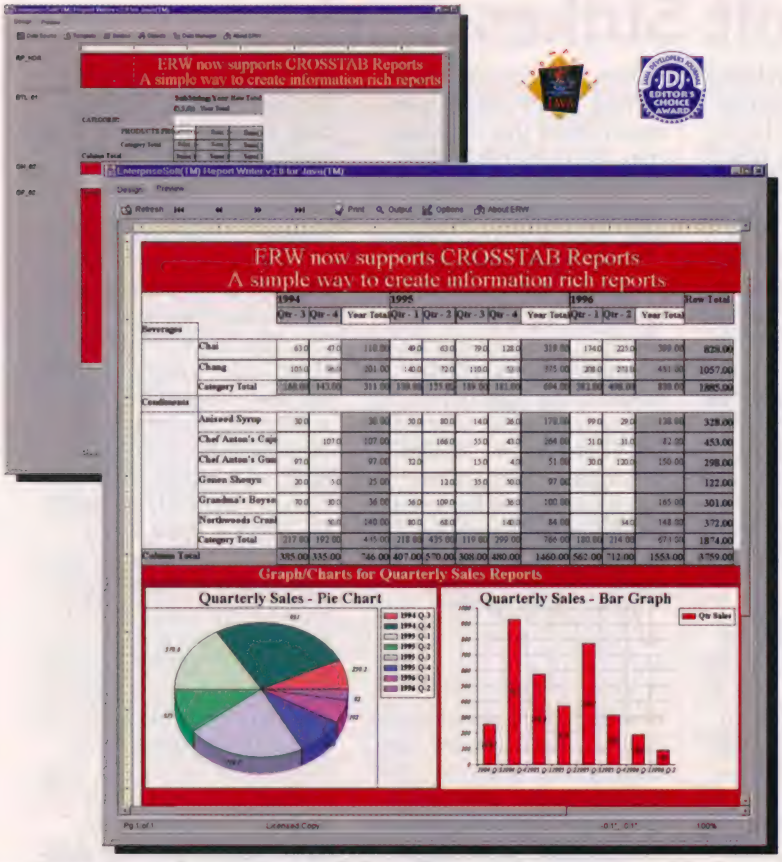
Sun recently launched a major initiative in the consumer arena with the establishment of its Consumer and Embedded, which is dedicated to developing technologies for the emerging markets using Jini and other Java technologies. Sun is also working with companies such as **America Online (www.aol.com)** to utilize a Jini technology-enabled network for the delivery of next-generation content and services to any device.

With Jini, Sun is also deploying a new licensing model. Sun will make Jini technology source code freely available through the Community Source License (CSL). This model enables developers to present the newest innovations and allows the marketplace to decide which innovations stay and what compatibility standards must be upheld. There is no cost for non-commercial use and it coexists with existing licenses. For commercial use, Sun expects Jini license royalties at 10 cents per device or $250,000 per product line.

Among publicly announced Jini licensees are: **Adaptive Networks (www.adaptivenetworks.com)**, America Online, **BEA Systems (www.beasys.com)**, **Bosch Siemens (www.siemens.de)**, **Bull (www.bull.com)**, **Canon (www.canon.com)**, **Cisco Systems, Inc. (www.cisco.com)**, **Computer Associates (www.cai.com)**, **Dallas Semiconductor (www.dalsemi.com)**, **Ericsson (www.ericsson.com)**, **Funai (www.funai.com.tw)**, **Inprise Corporation (www.inprise.com)**, **Kinko's Corporation (www.kinkos.com)**, **Kodak (www.kodak.com)**, **Metrowerks (www.metrowerks.com)**, **Motorola (www.mot.com)**, **Novell (www.novell.com)**, **Nokia (www.nokia.com)**, **Oki Electric (www.oki.com)**, **Philips (www.philips.com)** and **Sony Corporation (www.sony.com)**.

ENTERPRISESOFT™ INTRODUCES

Enterprise Reports™ 3.0


ERW now supports CROSSTAB Reports
A simple way to create information rich reports
ERW now supports CROSSTAB Reports
A simple way to create information rich reports
Graph/Charts for Quarterly Sales Reports
Quarterly Sales - Pie Chart
Quarterly Sales - Bar Graph

FEATURES

- Swing Based or AWT Based
- Report from Java™ Application Data or JDBC Data
- PDF, HTML, ASCII, Applet, Printer
- Crosstab Reports
- Form Reports
- Multiple Section Reports
- Parameterized Field Reports
- Graphs/Charts
- Easy-to-use Designer
- Table of Contents for Reports

Download a free 30-day evaluation copy www.EnterpriseSoft.com

WHAT CAN ERW DO FOR YOU ?

Extract ERW is data-source independent. It can report on data inside your Java™ application objects or from a JDBC/ODBC database. ERW implements a sophisticated object/relational mapping for Java™ application reporting which is fast and scalable to several thousand objects.

Analyze Analyze your data using sophisticated crosstabs, multiple sections, charts/graphs, table of contents, formulas as well as user-defined functions and query operators. The ERW Report Engine provides blazing fast, real-time report generation.

Format Use the familiar, easy-to-use ERW Report Designer to format the report structure and use the extensive customization features to define attributes of your report data.

Output Any ERW report can be exported to the PDF, HTML or ASCII file formats. In addition, the reports can be viewed on screen using the ERW Report Viewer applet or sent to a printer for a WYSIWYG hard copy.

THE FIRST REAL REPORTING/ PRINTING SOLUTION BUILT WITH YOU, THE JAVA™ DEVELOPER, IN MIND...

- 100% Pure Java™
- Limited, free redistribution of ERW run-time
- Small footprint (ERW run-time <350KB)
- Extendable, scalable
- Easy-to-use designer and API
- Extensive docs and sample code
- Works With Most Application Servers and Java™ IDEs
- Developer-based licensing: $495 for single license; $995 for annual subscription

Try Enterprise ReportS for yourself ! Find out why leading organizations such as NationsBank/BankAmerica, Deutsche Bank, EDS, Fedex, Ford Motor Co., IBM, US Navy, US Army, and The Getty Museum rely on ERW for their Java™ reporting/printing solutions! Download a free 30-day evaluation copy from our Web site today.

EnterpriseSoft™

7573 Waterford Drive, Cupertino, CA 95014 ■ PHONE 510-742-6700 ■ FAX 510-742-6800
■ EMAIL java@EnterpriseSoft.com ■ WEB www.EnterpriseSoft.com

EnterpriseSoft, the EnterpriseSoft logo, Enterprise Reports and "Enterprise-wide Business Intelligence Solutions" are trademarks of EnterpriseSoft. Java and Java related trademarks are property of Sun Microsystems, Inc. All other trademarks are property of their respective owners.

by Philip J. Gill javabuzz@sigs.com

Microsoft Strikes Back

Just 10 days before **Sun Microsystems Inc. (www.sun.com)** formally unveiled its Jini technology, **Microsoft Corp. (www.microsoft.com)** took the wraps off a technology meant to rival, if not preempt, Jini.

The technology, called Universal Plug and Play, is an extension of the Plug and Play technology that Microsoft introduced to the PC market in 1992, along with partners **Intel Corp. (www.intel.com)** and **Compaq Computer Corporation (www.compaq.com)**. With Plug and Play, a user could connect a new printer or scanner to a Windows PC without having to reboot the system to have it recognized.

Universal Plug and Play seeks to extend that to home networks and the Internet. Universal Plug and Play is a set of specifications for application programming interfaces (APIs) that will enable any Windows CE-based device to connect to a home network, the Internet, or an Intranet at any time, from anywhere, and be recognized by that network. Once recognized, it would be able to access network services and resources. Windows CE is a scaled-down version of the Windows platform, intended for handheld PCs and other low-end and, now, embedded devices. Conceptually, Universal Plug and Play will be used on any device that can house a chip, that is, everything from Webphones and handheld PCs, to television set-top boxes, even toasters and refrigerators.

If this sounds strikingly familiar to Sun's Jini, it should. Universal Plug and Play is Microsoft's attempt to pose an alternative that is entirely devoid of any Java content. Jini, as you might recall, is based on Sun's PersonalJava platform and written entirely in Java. Sun disclosed and has been testing Jini technology since last July, giving it a six-to-nine month lead on Microsoft.

Joining Microsoft in its Universal Plug and Play announcement were a who's who of the home PC world, including Intel, **Hewlett-Packard (www.hp.com)**, Compaq, **Dell (www.dell.com)**, **Cisco Systems (www.cisco.com)**, **Eastman Kodak (www.kodak.com)**, **Samsung (www.samsung.com)**, **Toshiba (www.toshiba.com)**, **Lucent (www.lucent.com)**, **Axis (www.axis.com)**, **3Com (www.3com.com)**, **Texas Instruments (www.ti.com)**, **NEC (www.nec-global.com)**, **Fujitsu (www.fujitsu.com)**, **Hitachi (www.hitachi.co.jp)**, and **AT&T (www.att.com)**.

Most expect the formal API specifications to be unveiled at Microsoft's annual technical conference for hardware developers, the Windows Hardware Engineering Conference, April 7th through 9th at the Los Angeles Convention Center.

It is expected that the first Microsoft product to embody the Universal Plug and Play API specifications will be Windows CE 3.0, which is expected to ship mid-year. However, over time, all Microsoft operating systems, including Windows 2000, are expected to include some or all Universal Plug and Play capabilities. There is also a planned upgrade for Windows 98 users.

According to **Dwight Davis**, service director for the Microsoft as a Partner and Competitor Service of Summit Strategies Inc., Microsoft and its Universal Plug and Play partners—among them, Sun enterprise server rival Hewlett-Packard Co.—are positioning Universal Plug and Play as an "open standards"-based initiative for the embedded Internet appliances. By that, said Davis, Microsoft intends to base its specifications on the use of widely available Web-based technologies, primarily, TCP/IP, HTTP, HTML, and XML.

In contrast, Davis said

continued on page 13 ➤➤

Sun Beats Back Rival Embedded Java Standard

Sun Microsystems Inc. (www.sun.com) has beaten back, at least for now, an attempt to create a rival Java standard to its own PersonalJava specification in the embedded systems market. The rival proposal, backed by **Hewlett-Packard Co, (HP; www.hp.com)** and the HP-led **Real-Time Java Working Group (RTJWG; www.newmonics.com/webroot/rtjwg.html)** has been rejected by the **National Committee for Information Technology Standards (NCITS: www. ncits.org)**. The HP-led RTJWG also includes **Microsoft Corp. (www.microsoft.com)** and **NewMonics, Inc. (www.newmonics.com)**. It was formed in October.

Members of the Real-Time Java Working Group still have concerns about Sun's overall control of the development process.

The HP-led group late last year had elected to submit a proposal for a real-time Java extension to NCITS for consideration, and NCITS' initial vote supported that effort. Ten NCITS members voted in the affirmative—enough to pass the measure.

But negative votes from Sun, **IBM (www.ibm.com)**, and **Sybase Inc. (www.sybase.com)** triggered a 10-day reconsideration period. IBM is said to have raised concerns about the potentially harmful effects of having multiple development efforts, and Sybase cited potential infringements on Sun's intellectual property rights.

In the end, seven NCITS members voted in favor of the proposal, six voted no, five abstained, and one didn't indicate a choice.

➤➤ *continued from page 10*

security products for Internet applications.

Companies announcing new membership in the JSA today include **NEC USA Inc. (www.nec.com)**, **Network Appliance, Inc. (www.networkappliance.com)**, **Valicert Inc. (www.valicert.com)**, and **Fortress Technologies Inc. (www.fortresstech.com)**. They join more than 25 companies already supporting mobile code inspection as a routine security requirement for any enterprise using the Internet.

InstallAnywhere Goes Global

Zero G Software, Inc. (www.ZeroG.com) of San Francisco has brought out localized versions of its InstallAnywhere product. Designed to aid software developers in their efforts to deploy software outside of the United States, InstallAnywhere's International Language Packs allow for the creation of Java-based installers in 29 different languages.

To assist software developers in their global deployment efforts, Zero G is offering the Core Language Pack to all users of InstallAnywhere 2.5 Standard Edition for free. The languages included in the Core Language Pack are English, French, German, Italian, Japanese, and Spanish.

In addition to the Core Languages Pack, two separate Language Packs are also available, Western and Eastern: The Western Language Pack includes Basque, Catalan, Danish, Dutch, Finnish, French (Canadian), Indonesian, Norwegian, Portuguese (Iberian), Portuguese (Brazilian), and Swedish.

The Eastern Language Pack includes Chinese (simplified), Chinese (traditional), Czech, Greek, Hungarian, Korean, Polish, Russian, Slovak, Slovenian, Thai, and Turkish.

Microsoft Strikes Back *continued from page 12*

Microsoft and its partners will contend that Sun's Jini is proprietary, being that it is based entirely on technology that Sun owns and controls. Of course, Microsoft and its partners won't play up the fact that the Universal Plug and Play specifications require hardware and software developers to write to Microsoft's own proprietary Windows CE technology.

Microsoft is also expected to argue that Windows CE, soon to be released in its third iteration, is a more proven and reliable technology than Java and Jini. They might also argue that CE, therefore, also has greater developer support and a greater number of device drivers for critical peripherals, than Java and Jini.

Sun isn't going to roll over and let Microsoft dominate with this critical space. Sun already has a substantial time lead, and in the Web, first out often means first across the finish line. Microsoft is definitely in a come-from-behind mode. Therefore, it won't surprise many analysts if Sun, in partnership with one or more of its Jini partners, is first to deliver real Jini-based products, while Microsoft talks about specifications and show prototypes and mock ups. At press time, Sun was expected to make an announcement in late January, revealing the first Jini manufacturers. That will further Sun's lead.

It's also important to note that many of Universal Plug and Play's initial backers have already signed on to develop Java products in this embedded Internet appliance space as well. **Toshiba (www.toshiba.com)** is one, for instance, that is developing a Jini and PersonalJava product. IBM is a major backer and developer of Java technologies. And HP, although it's backing Jini, it's not backing PersonalJava, Sun's Java specification for the embedded systems market. HP, instead, is developing products around its own Java Virtual Machine (JVM), the ChaiVM. Many manufacturers are likely to play both sides of the market for sometime to come.

The Microsoft empire may have fired back its first round, but the battle for the embedded Internet appliance space is far from over.

BEA and Object People Ally for Rapid Java Development

BEA Systems, Inc. (www.beasys.com) and **The Object People (www.objectpeople.com)** have formed an alliance to integrate TOPLink for Java, an object-to-relational mapping product, and BEA WebLogic, a Java-based Web application server. The integration is designed to help enable customers to efficiently build components for application servers that run on Java, while significantly cutting application development time and expense.

Additionally, the integration should help enable customers to focus on the business requirements of their applications, not on the development of complex infrastructure that would otherwise be required to support interactions with relational data.

The Object People, with cooperation from BEA, have also developed TOPLink for BEA WebLogic, a new product that is fully integrated with the BEA WebLogic application server. This product should assist customers in leveraging their database investments, while implementing applications that incorporate advances in modern computing technology to meet their business needs.

W3C Releases New Java Web Server

The **World Wide Web Consortium (W3C; www.w3c.org)** has released an open source version of Jigsaw, the award-winning Java Web server it developed for experimenting with new server technologies. Jigsaw version 2.0 provides faster Web access through the use of HTTP/1.1 and an open source implementation of the Java servlet interface.

Jigsaw is implemented in Java, so will run on most platforms out of the box. Jigsaw 2.0 has been tested on Windows 95, Windows NT, and Solaris 2.x. Successful installations on OS/2, MacOS, BeOS, and AIX have also been reported.

javabuzz art: p10, Jim Frazier/SIS.

Sun Gains HAVi Backers in Java Battle

In what can only be seen as a major victory over **Hewlett-Packard Co. (HP; www.hp.com)**, **Microsoft Corp. (www.microsoft.com)**, and other Java challengers in the consumer electronics market, **Sun Microsystems Inc. (www.sun.com)** has won the support of key international consumer electronics companies involved in the digital television industry for its Java TV application programming interface (API) specification.

Backers of Sun's Java TV API include **Hongkong Telecom (www.hkt.net)**, **LG Electronics (www.lge.co.kr)**, **Matsushita (www.panasonic.co.jp)**, **Motorola (www.mot.com)**, **OpenTV (www.opentv.com)**, **Philips (www.philips.com)**, **Sony (www.sony.com)**, and **Toshiba (www.toshiba.com)**. These companies have said they will cooperate with Sun in the further development of the Java TV API.

This API will extend the Java platform to enable the secure delivery of interactive television content and services. This working group, including Sun and the above-named industry leaders, is developing the Java TV API to be incorporated into global digital television standards. The first draft of this API was expected to be released for public review by end of the first quarter of this year.

A set-top box or digital television that supports the Java TV API will offer consumers interactive television content such as enhanced television, Video-On-Demand (VOD), Electronic Programming Guides (EPGs), and interactive multi-camera-angle sporting events. The API will also address other functions, such as audio/video streaming, conditional access, and access to in-band and out-of-band data channels.

In addition, the Java TV API will provide an independent software platform to access the hardware features that are unique to televisions, such as tuner control for channel changing and on-screen graphics, while maintaining portability across operating systems and microprocessors.

The Java TV API is intended to address the needs of manufacturers, system operators, and content providers as they seek standards to deliver interactive content securely to consumers via cable set-top boxes, satellite receivers, and digital televisions.

Alcatel Demos Java-Based Screenphone

Alcatel (www.alcatel.com) demonstrated one of the first Java-based Internet screenphones at the Consumer Electronic Show (CES'99) in Las Vegas.

Alcatel's Internet screenphone is a next-generation compact telephone with built-in Internet access which combines a keypad, a color VGA screen, a touch screen, and a smart card reader. It provides easy and affordable access to the Internet as well as advanced telephone services. It facilitates short transactions and frequent consultations of Internet content, as well as email and voice communications.

Alcatel's screenphone is fully compliant with the Internet Screenphone Reference Forum (ISRF) specifications and is based on the JavaOS from Consumers from **Sun Microsystems Inc. (www.sun.com)**. The ISRF, formed to define worldwide Internet screenphone standards, is supported by more than 20 major companies from the IT community.

Take the Java Challenge

Do you have what it takes to be the Java Programmer of the Year? Take the Java Challenge and see how your skills stand up against the industry's best developers. Sponsored by RogueWave, the Java Challenge will be a series of Java software development questions prepared by Editor Dwight Deugo. Your "challenge" will be to answer as many questions as you can within 10 minutes.

Object Expo New York/SIGS Expo for Java
New York Marriott World Trade Center
Conference, March 28-April 1, 1999
Exhibition March 30-31, 1999

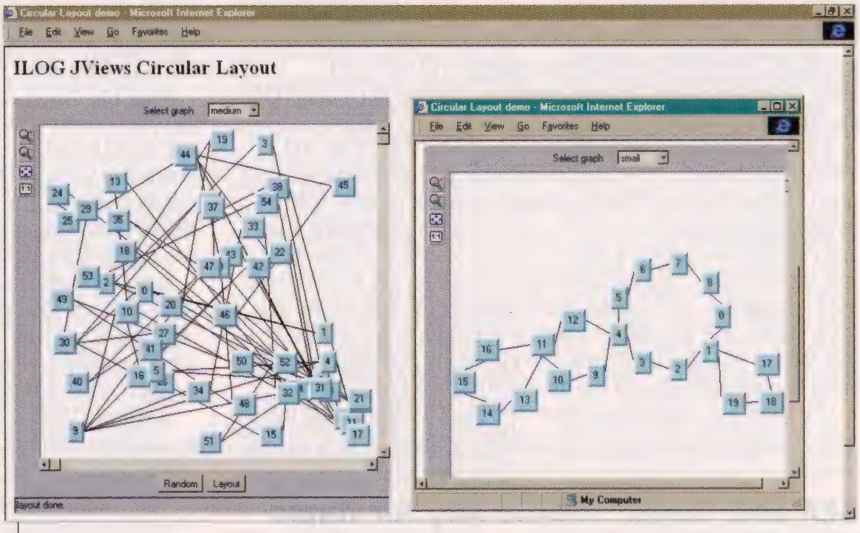
Class Libraries That Integrate Into the Existing AWT

JViews encourages 2D developers to take Java plunge

Reviewed by Todd C. Zino/NSTL

Version Reviewed: 2.0
Current Version: 2.0

Cup rating system:
4 Outstanding **3** Good **2** Acceptable **1** Poor



Using the JViews graph layout algorithms, nodes can be arranged randomly (left) or in compounded geometric structures (right).

JAVA'S ADVANTAGES IN designing applet user interfaces has earmarked it increasingly for use in two-dimensional data plotting. With this in mind, ILOG has developed JViews as a port of its longstanding C++ Views package. The company has long pioneered libraries that offer graphics and vector wizardry. The idea is to provide functionality for handling map and graph data, prototyping arrangements of graphics in precise form, and binding to underlying data. This is presented to Java developers as a system of class libraries that integrate into the existing AWT.

Unlike most class packages of its nature that provide rich display of tabular data, JViews is more concerned with the arrangement and viewing of complex data—primarily maps and node graphs with heavy interrelations. Each package bears numerous sorting algorithms and utility classes derived from academic lecture halls. ILOG classifies its package into three layers, ascending from the AWT foundation: graphic behavior and viewability, "manager" objects for handling dynamic views on visual data, and a grapher for defining and manipulating relationships among "nodes."

ILOG JViews installs easily as a zipped class file and has no user interface of its own. It is designed purely to be an API with binary class libraries. There is one utility, ILVEdit, for visually editing the auxiliary ".ILV" files JViews uses to store imaging coordinate information.

As typical with the newer breed of Java third-party APIs, the documentation and reference was meager and unable to fully define each class's place and ideal use in a non-trivial scenario. Luckily, the example demonstration applets were well-written and yielded more insight into using multiple JViews classes together over an AWT canvas.

The first level examined was the Graphic Viewing classes. This required setting up a `GraphicBag` instance into which graphics can be placed, dynamically zoomed and resized, and bound to user-defined properties. The `GraphicBag` bears no resemblance to the infamous `GridBagLayout` class from AWT but rather carries its own rich coordinate system. The main goal of the JViews Manager objects is to store abstract relational metrics for each item placed. In this fashion, the actual display can be set to different zoom levels and objects manipulated in more convenient relational ways than simply changing their hard-wired coordinates. Such operations include shrinking objects on the fly, retrieving the smallest "bounding box" around an arbitrarily shaped image, and rotating the object.

The cartography libraries for JViews are easily the most advanced in the package, and tailored for anyone who wishes to handle dynamic Geographical Imaging Systems (GIS) data in a more thread-safe language like Java. Maps handled in JViews can be raster- or vector-based and imported from many of the industry standard formats (e.g., OGDI, CADRG, DXF, SHAPE, DTED).

The JViews representation of map data extends all the way to the basic point, line, polygon, and vector infor-

The impossible deadline:

Deliver web access to the sales database by NOON TODAY.

No problem.

Meet the challenge with VisualCafé Version 3:
The fastest Java™ tool for delivering database information.

Need SQL via JDBC through HTTP and Servlets? Build it today with VisualCafé Version 3 Database Edition– the right tool to create complex, web-enabled transaction oriented database applications.

Powerful new database functionality accelerates application development:

- **Open JDBC development environment**
- **Instant Query By Example (QBE)**
- **SQL and stored procedure beans**
- **Validation and calculation rules**
- **Full support for JFC/Swing components and data model**
- **Database wizards including master detail support**

CHECK OUT OUR SPECIAL DEVELOPER OFFER

at www.visualcafe.com/vc3/jroffer

For information on site licensing, call Symantec Corporate Sales at 1-888-822-3409

NEW! Powerful Distributed Debugging
For a preview of VisualCafé Enterprise Suite, see www.visualcafe.com/vces/walkthrough

Computer Discount Warehouse	J&R Computer World	MicroWAREHOUSE	PC Connection	PROVANTAGE	Programmer's Paradise
1-800-903-4239	1-800-221-8180	1-800-397-8508	1-800-800-4656	1-800-336-1166	1-800-445-7899

©1999 Symantec Corporation. All rights reserved. Symantec and the Symantec logo are registered trademarks of Symantec Corporation in the United States. VisualCafé is a trademark of Symantec Corporation. Oracle is a registered trademark of Oracle Corporation. Java and all Java based trademarks and logos are trademarks or registered trademarks of Sun Microsystems in the U.S. and other countries. In Canada, call 1-800-365-8641. In Australia, call 02-9850-1000. In Europe, call 31-71-535-3111.

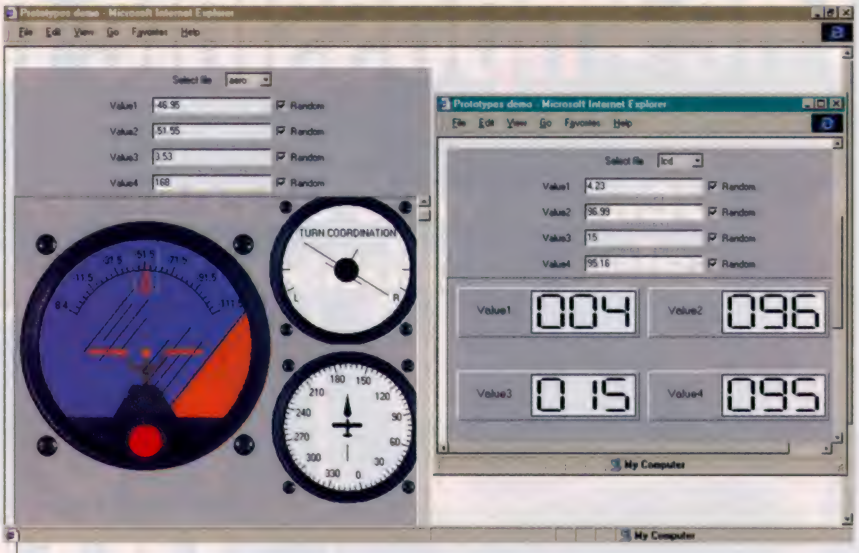
The Fastest Java Development Solution For Corporate Databases.

SYMANTEC.

mation. This grants the developer the ability to perform all of the standard zoom, add, and change operations on existing maps generated by other applications. The JViews map package includes operations to render the map data in many projections (e.g., cylindrical, conic, planar) once the underlying vector data is instantiated. When used in tandem with the Manager classes, each component of map data can be assigned its own listener to filter incoming thread events. Using this functionality, it is possible to restrict "redrawing" to only designated areas when data is updated.

In keeping with the current trends in object-oriented technology, JViews includes components for binding graphic objects to business logic. These templates, designated as "prototypes" by ILOG, are roughly akin to the C++ template classes except bound to visual coordinate/polygon structures created by the developer. It was extremely easy to create simple geometric designs with the aforementioned ILV Editor, but alas, near impossible to do anything non-trivial. This was perhaps the biggest disappointment with any one aspect of the current JViews bundle. The utility had a very arcane interface, and the documentation included was not enough to demystify the advanced techniques needed to create graph plots. It also claimed to be capable of generating ILV files from standard .gif images, but this was impossible to actually carry to fruition. It would be really nice to have a simple method of importing graphics into the editor and generating coordinate plots and geometric logic rules from this.

In addition to the mapping and prototyping classes, JViews also provides extensive methods for laying out items in graph representation. This use of the term "graph" refers to the computer science definition of arbitrarily arranged and interconnected nodes in space, not the business charts and spreadsheet views generally connotated by the word. The most common uses of abstract graphs are to represent tree structures and node topologies (particularly electrical wiring and network layout). The implementation is fairly straightforward: once a graph has been declared with the grapher and its nodes added to the data structure, it can be laid out manually (i.e., explicitly placing each node at a certain point) or automatically (employing included or newly defined algorithms). Among the predefined algorithms are many optimizations of classic mathematical forms, such as Topological Mesh, Spring Embedder, and Uniform Length Edges.



Intricate geometric shapes with logical properties can be created from scratch using JViews' ILV Editor and prototyping libraries.

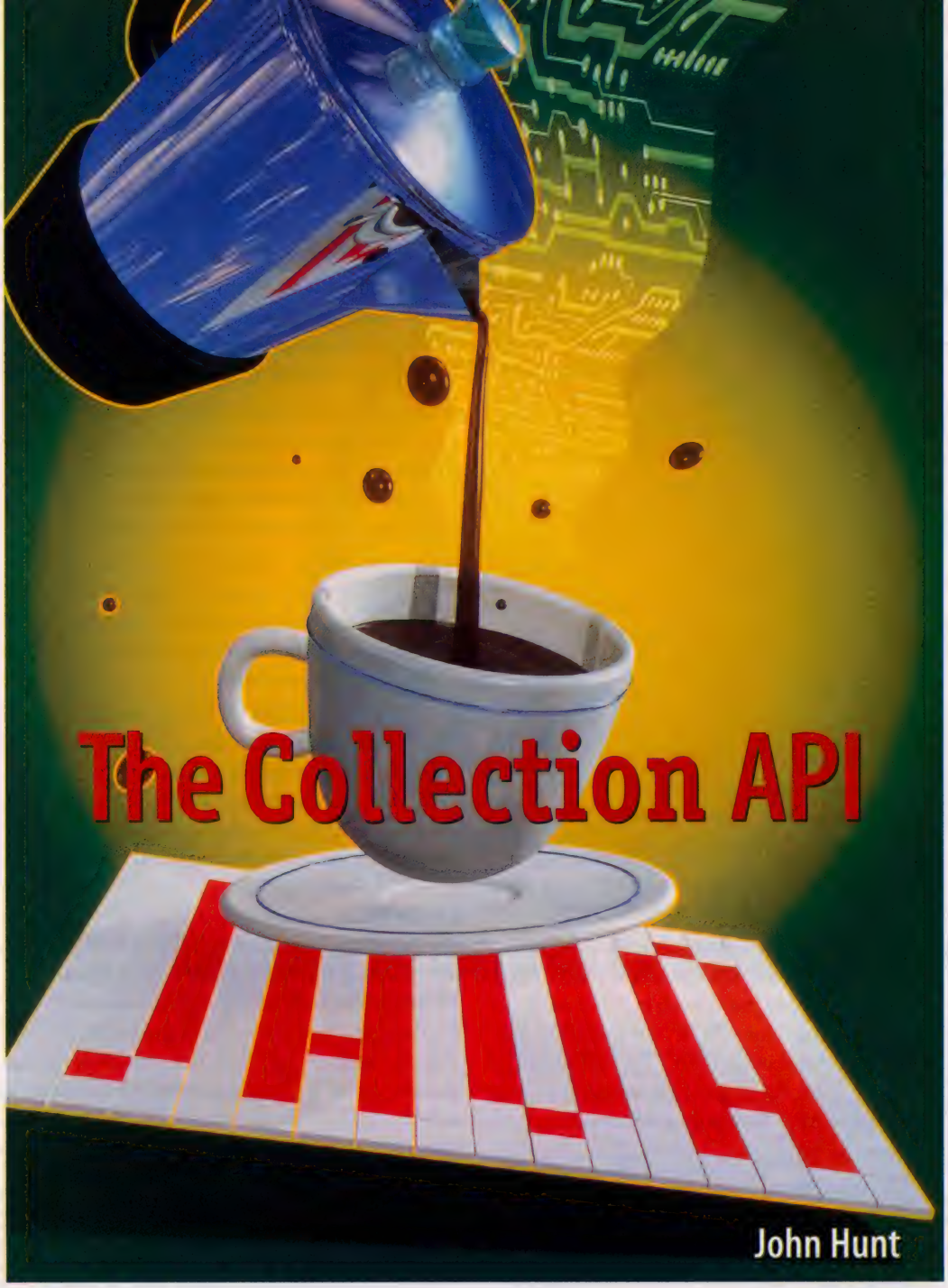
Most of JViews' limitations are confined to usability quirks. A lot of its old-school "C++ visual library" feel is still present, resulting in many high learning curves and tedious processes to completing tasks. Nonetheless, these are tasks that would be even more tedious—if not entirely impossible—without JViews. While easily clinching its status as a forerunner in map and graph plotting, JViews will meet with competition in the area of business prototyping if it is not able to improve the versatility of importing and designing usable graphics. Because JViews is essentially a port from C++—an environment where there is no de facto visual layout standard—it will probably take a few subsequent releases for it to integrate more clearly with existing AWT and Bean technology. The JViews classes can be "beanified" manually for use in Visual Java IDEs, but this should be a more automatic process.

As it currently stands, JViews excels at what it does without getting in the way of other technologies a developer might wish to integrate concurrently in a large development path. However, its feature-set might be overkill if you're not specifically looking for Web-based GIS, interactive data-driven animations, or node graphing solutions. There are simpler and cheaper methods for the very basic layout and data binding, many of which are part of the improved JDK 2.0, but they may prove insufficient and inefficient for large-scale projects. JViews is certainly in a leading position to encourage high-end 2D developers (who have no doubt been eyeing Java curiously for a long time) to take the plunge. ■

Todd C. Zino works in publications and development at National Software Testing Laboratories. In addition to reviewing various object-oriented development tools, he is involved with programming and testing of network environments. He can be contacted at todd@lacemaker.com.

Vendor Info
ILOG, Inc.
1901 Landings Drive
Mountain View, CA 94043
800.367.4564
www.ilog.com

Quantity reprints of this article can be purchased by phone: 717.560.2001, ext.39 or by email: sales@rmsreprints.com.



Roy Wiemann/Image Bank

The Collection API interface and classes will be the cornerstone of most implementations

JDK 1.2 INTRODUCED A new set of collection classes. A collection, according to JavaSoft, is a single object representing a group of objects. They are a *collection* of other objects. Collections may also be referred to as *containers* (as they contain other objects). These collection classes are used as the basis for data structures.

The new collection classes extended the facilities provided by the utility classes Vector and Hashtable. These classes are called by names such as HashSet, ArraySet, LinkedList, etc. Some of the collection classes, for example, ArrayList, provide functionality similar to existing data structure classes such as the Vector class but have been optimized for performance. Like Vector and Hashtable the collection classes can only hold objects, thus if you wish to hold the basic types within them, you need to wrap the basic type in the appropriate object wrapper. The Collections API is part of the java.util package.

WHAT IS IN THE COLLECTION API?

The collection classes and interfaces in JDK 1.2 are collectively referred to as the JDK Collections API. A collection is a group of objects (these objects are called the elements of the collection). Collections are the Java mechanism for building data structures of various sorts.

Interface Collection is the root of all collec-

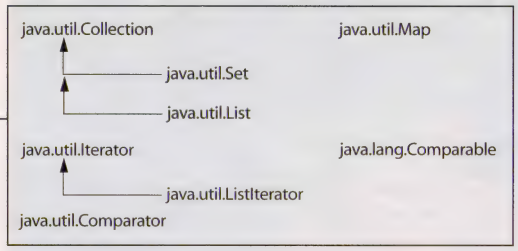


Figure 1.

The interfaces defined in the Collections API.

tions in the API. Figure 1 summarizes the collection API in Java. Some of the classes illustrated are abstract classes on which others build. In fact, the collection class hierarchy is a classic example of the use of interfaces and abstract classes and how they can be used to group together functionality as well as indicate what is expected of subclasses. The Collection API can be divided in the following manner:

- *Collection Interfaces:* Four collection interfaces represent the core types of collections supported by the Collections API. These interfaces are Collection, Set, List, and Map.
- *Abstract Implementations*: These provide the abstract root classes for the collection hierarchy. They specify the behavior which must be implemented by subclasses that provide the features of a Set, a List, etc.
- *Concrete Implementations*: General-purpose implementations of the collection interfaces.
- *The Collections Class*: This class provides a range of static factory methods. They are intended to enable pre-Collection API data structures to be converted into collections efficiently and effectively. This allows legacy APIs that return arrays to inter-operate with new APIs that expect Collections.
- *Iteration:* This is the new iteration mechanism introduced in as part of the JDK 1.2.
- *Array Sorting and Searching:* The Arrays class static methods for sorting and searching arrays are not really part of the Collections API. However, they are provided in JDK 1.2 for the first time, and they are useful in manipulating array-based data structures.

Just as with the older classes, Vector and Hashtable, the basic operations that are performed by the collection data structures include adding and removing elements, determining the size of the collection, querying the presence or absence of elements, and iterating over the elements.

COLLECTION INTERFACES

The interfaces that comprise the collection hierarchy are designed to allow manipulation of collections in an implementation-independent fashion. This allows for interoperability among unrelated APIs and applications that take collections as input or return them as output. This reduces the effort required to design, implement, maintain, and understand such APIs and applications. The interfaces that form the basis of the Collections API are:

- Collection: the root of most of the Collections API. The only classes that do not implement this interface are the map-oriented classes (Hashtable, HashMap, ArrayMap, and TreeMap).
- Set: represents the set data type. Sets contain all the elements put into it, in any order. Duplicates are not kept; adding equal objects many times results in only one such object in the Set; extends the Collection interface.
- List: defines an Ordered Collection data type (also known as a sequence). An ordered collection is where the ordering is given by the order in which elements were added; extends the Collection interface.
- Map: an association between pairs of objects. One object often acts the key to the other object (referred to as the value).

If you wish to define your own collection classes, then implementing one of these interfaces could be your starting point.

Collection

The Collection interface acts as the root of (almost) all collection classes. It defines the methods that all collections (except Map type collections) must implement. Note that some of the implementing classes may decide that the implementation should throw an UnsupportedOperationException to indicate that they should not be used with this type of collection.

The interface defines methods for adding an element to a collection, accessing elements in a collection, removing an element from a collection, and indicating the size of the collection. It also provides facilities that allow an iterator to be returned (these are similar to enumerations in JDK 1.1). The toArray() method also allows the collection to be converted into an array.

Any object can be stored into any collection (again, implementations of a particular collection may limit the type of object they hold). This means that a collection can be a very flexible way of holding other objects. Typically, collections are used as data structures, however, they can also be used as temporary holding places for groups of calculations, results, classes, etc.

The Collection interface includes:

- add(Object) and addAll(Collection): Ensure that this Collection contains the specified element or elements (optional operations).

Formula One™

Technology for delivering Spreadsheets Everywhere.™

Java.
Web pages.
Multiple platforms.
Web-based applications.
Server-side functionality.
Spreadsheets.

Formula One for Java ties them all together, empowering you to deliver spreadsheets everywhere.

With Formula One for Java, Java developers are immediately armed with a powerful spreadsheet JavaBean for use in environments such as Visual Café and JBuilder.

Use it on a server to calculate and format data. Use it in Internet or intranet applications. Connect it to databases. Or read and write Excel files.

The same fast calculations and robust spreadsheet functionality is also available to Webmasters, who can use it to add live spreadsheets to their pages with as few as one line of HTML code.

Even end users on any desktop can use it as a standalone application.

Thousands of businesses such as State Street Bank and Simon & Schuster trust Formula One in their applications.

You, too, can leverage superior spreadsheet functionality and Java in your next application.

Tie them together.

Deliver spreadsheets everywhere.

Use Formula One for Java, the Internet Spreadsheet™.

Formula One™ Professional

- ✓ Formula One for Java® 5.5.1
  - ✓ Application, applet, and JavaBean
  - ✓ Read/write Excel files
  - ✓ 100% Pure Java
- ✓ Formula One ActiveX™ 6.0
  - ✓ Read/write Excel files
  - ✓ First Impression ActiveX 6.0 charts
- ✓ Fast calculations
- ✓ Excel formula and function syntax
- ✓ Workbook designer
- ✓ Robust API
- ✓ Database connectivity
- ✓ Small footprint and downloads
- ✓ Server and client side functionality
- ✓ Write spreadsheets to HTML
- ✓ Per user and server/CPU pricing

Tidestone TECHNOLOGIES™

INFORMATION: **800-922-9665** • **FREE FULL-FEATURED TRIALS:** **www.tidestone.com/ssheets**

Copyright © 1999 Tidestone Technologies, Inc. All rights reserved. Tidestone, the Tidestone logo, Formula One, Spreadsheets Everywhere, Internet Spreadsheet, and First Impression are trademarks of Tidestone Technologies, Inc. Java, 100% Pure Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All other trademarks are property of their respective owners. Specifications subject to change without notice.

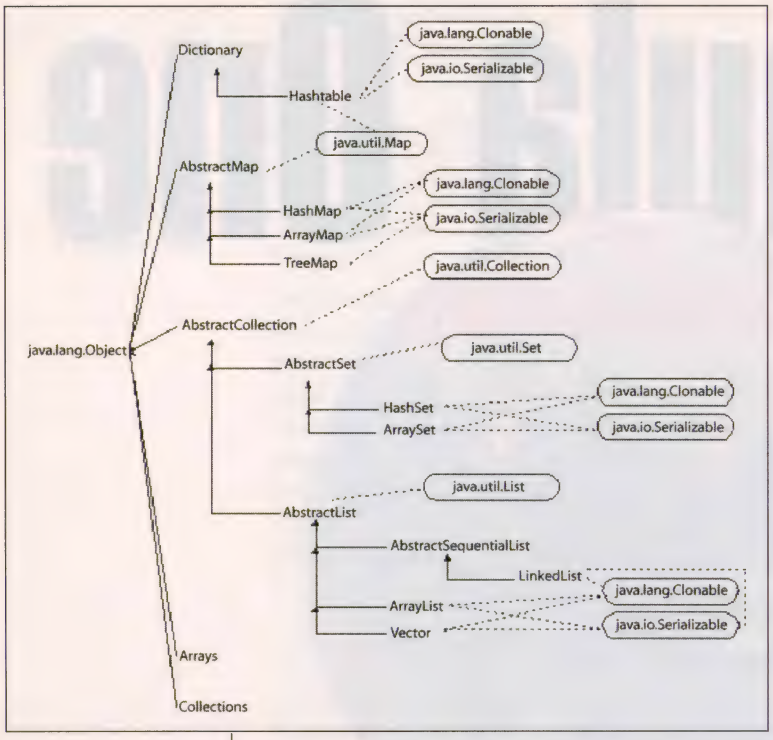


Figure 2.

The Collection API class hierarchy.

- clear(): Removes all of the elements from this Collection (optional operation).
- contains(Object) and containsAll(Collection): Returns true if this Collection contains the specified element (or elements).
- equals(Object): Compares the specified Object with this Collection for equality.
- isEmpty(): Returns true if this Collection contains no elements.
- iterator(): Returns an Iterator over the elements in this Collection.
- remove(Object) and removeAll(Collection): Remove an instance of specified elements from this Collection, if it is present (optional operations).
- retainAll(Collection): Retains only the elements in this Collection that are contained in the specified Collection (optional operation).
- size(): This returns the number of elements currently held in the Collection.
- toArray(): The result of this method is an array version of the Collection.

Some of these methods are labeled "optional." This denotes that some implementations may not perform one or more of these operations. If they are called, they should throw a runtime exception, the aforementioned UnsupportedOperationException. This means that collection implementations must specify in their documentation which optional operations they support. JavaSoft has introduced several terms to aid in this specification:

- Collections that do not support any modification operations (e.g., add, remove, clear) are referred to as unmodifiable. Collections that are not unmodifiable are referred to as modifiable.
- Collections that additionally guarantee that no change in the Collection will ever be observable via "query" operations (e.g., iterator, size, contains) are referred to as immutable. Collections that are not immutable are referred to as mutable.
- Lists that guarantee that their size will remain constant even though the elements may change are referred to as fixed-size. Lists that are not fixed-size are referred to as variable-size.

It is up to the individual implementation whether it wishes to limit the type of object that it holds or not. If a programmer attempts to add an element that violates an implementation's restrictions, then that collection should generate a runtime exception. Under normal circumstances this should be one of ClassCastException, IllegalArgumentException, or NullPointerException.

JavaSoft also states that all general-purpose Collection implementation classes should provide two "standard" constructors: a void (no arguments) constructor, which creates an empty Collection, and a constructor with a single argument of type Collection, which creates a new Collection with the same elements as its argument.

There is no way to enforce these recommendations (as interfaces cannot contain constructors), but all of the general-purpose Collection and Map implementations in the JDK 1.2 comply with this guideline.

Set

The interface Set is basically the same as the interface Collection, with the exception that it does not allow duplicates.

List

A list is an ordered collection of elements. It has a very specific sequence to the elements it contains. That order is determined by the order in which objects are added to the ordered collection/List instance. An implementation of the List interface can hold any type of object and can be used in situations where the order in which the objects were added to the instance must be preserved. In general, List implementations will allow duplicate objects.

There is a range of order-related messages that allow objects to be added and accessed with reference to the order in the List instance. For example, it is possible to access an object at a particular location, to find the position of an object, or the last position of an object using methods such as get(int), set(int Object), indexOf(Object), indexOf(Object, int), and lastIndexOf(Object). Note

THEY SHIP IT WITH A PRIZE IN EVERY BOX.

Inside every box of the world's most popular development suite is the power of the world's leading visual modeling technology.

"By working together to closely integrate modeling and development tools, Microsoft and Rational have made modeling more accessible to all Windows developers. Microsoft Visual Modeler, developed from Rational Rose technology, is now included in every box of Visual Studio. This partnership provides a great way for developers to take advantage of the full functionality which Visual Studio provides."

Paul Gross
Vice President Development Tools Division, Microsoft

RATIONAL SOFTWARE

UNIFYING SOFTWARE TEAMS

Microsoft Enterprise Development Partner

www.rational.com/mvs/vismod/ • 1-800-728-1212

Rational, the Rational logo, and Rational Rose are trademarks or registered trademarks of Rational Software Corporation in the United States and in other countries. Visual Studio 6.0 is a trademark or registered trademark of Microsoft Corporation. ©1998 Rational Software Corporation. All rights reserved.

that like arrays and Vectors, Lists are zero-based, thus the first location in a List is position zero. In addition, this interface inherits all the methods defined in the Collections interface.

To process Lists it would certainly be possible to iterate over the elements of the List using a standard for loop and the get(int) method. However, the time taken to access a particular object is linear with respect to its position in the list. It is therefore more efficient to use one of the Iterator access methods provided by a List. List provides a special Iterator, called a ListIterator that allows element insertion and replacement, and bi-directional access in addition to the normal Iterator operations. Two Iterator access methods are provided. One returns an iterator for the whole List, while the second returns an iterator that starts at a specified position in the List.

The methods defined in this interface (in addition to those inherited from Collection) are:

- addAll(int, Collection): Inserts all elements in the specified Collection into this List at the specified position (optional operation).
- get(int): Returns the element at the specified position in this List.
- indexOf(Object) and indexOf(Object, int): Return the index in this List of the first occurrence (after the optional specified position) of the specified element, or -1 if the List does not contain this element.
- lastIndexOf(Object) and lastIndexOf(Object, int): Return the index in this List of the last occurrence of the specified element (after the optional specified position), or -1 if the List does not contain this element.
- listIterator() and listIterator(int): Returns a ListIterator of the elements in this List (in proper sequence). The optional position indicates the start point for this list.
- remove(int) and removeRange(int, int): Remove the element at the specified position in this List or the sub-range between fromIndex, inclusive and toIndex, exclusive (optional operation).
- set(int, Object): Replaces the element at the specified position in this List with the specified element (optional operation).

Map

A Map is a set of associations each representing a key-value pair. The elements in a Map may be unordered but each has a definite name or *key*. Although the values may be duplicated, keys cannot. In turn, a key can *map* to, at most, one value. Some Map implementations, like TreeMap and ArrayMap, make specific guarantees as to their order—others, like HashMap, do not.

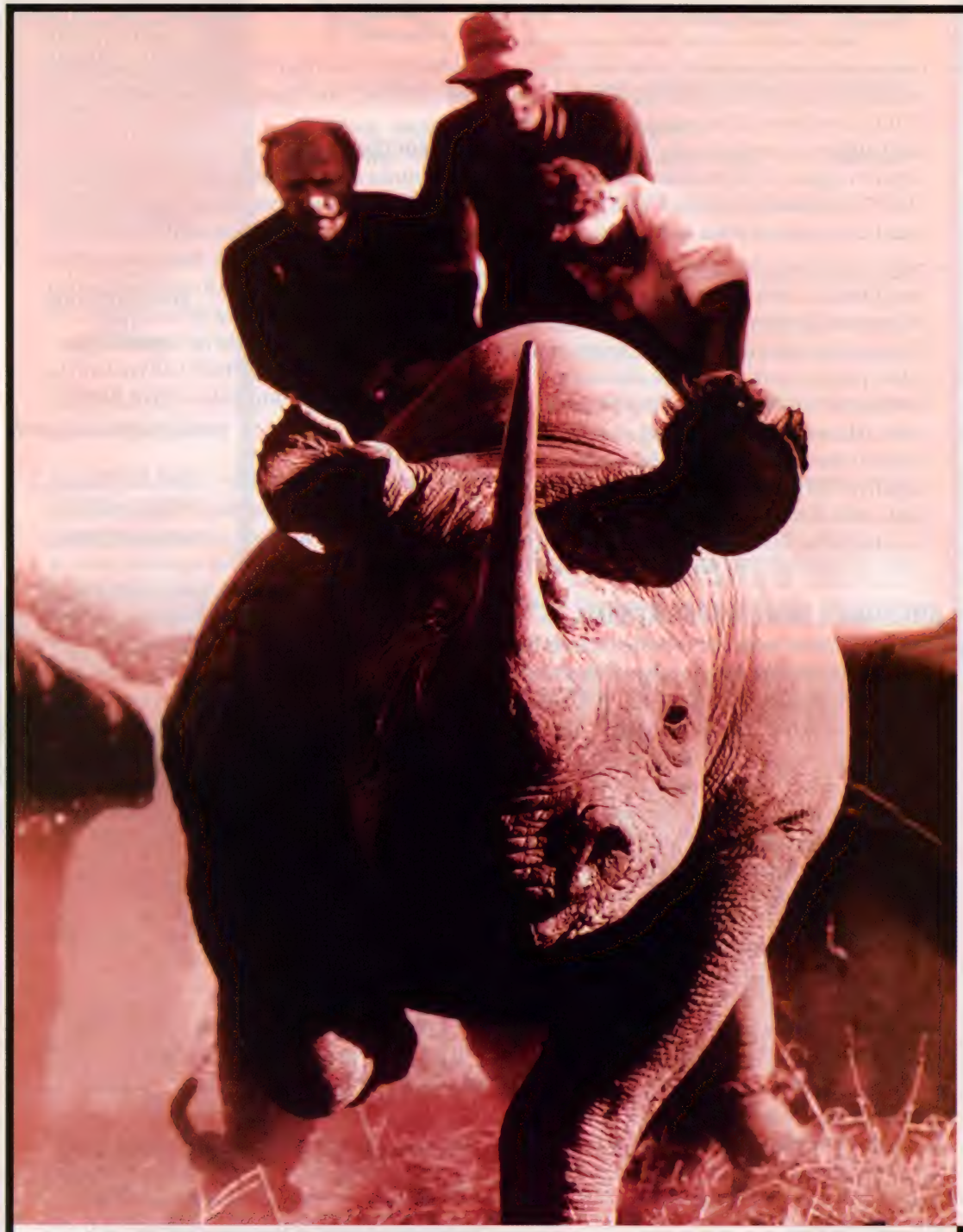
The Map protocol allows either the keys to be viewed, the values to be viewed, or for the keys to be used to access the values. JavaSoft has stated that all general-purpose Map implementation classes should provide two "standard" constructors: A void (no arguments) constructor, which creates an empty Map, and a constructor with a single argument of type Map, which creates a new Map with the same key-value mappings as its argument. In effect, the latter constructor allows the user to copy any Map, producing an equivalent Map of the desired class.

- clear(): Removes all mappings from this Map (optional operation).
- containsKey(Object): Returns true if this Map contains a mapping for the specified key.
- containsValue(Object): Returns true if this Map maps one or more keys to the specified value.
- entries(): Returns a Collection view of the mappings contained in this Map.
- equals(Object): Compares the specified Object with this Map for equality.
- get(Object): Returns the value associated with aKey.
- hashCode(): Returns the hash code value for this Map.
- isEmpty(): Returns true if this Map contains no key-value mappings.
- keySet(): Answers with a Set of keys from the receiver.
- put(Object key, Object value): Puts value into a dictionary with the external key.
- putAll(Map): Copies all of the mappings from the specified Map to this Map (optional operation).
- remove(Object): Removes the mapping for this key from this Map if present (optional operation).
- size(): Returns the number of key-value mappings in this Map.
- values(): Answers with a Collection of the values in the receiver.

COMPARISONS

To sort a collection, the objects within the collection either need to be Comparable or they need to be able to be compared using a Comparator. Comparable and Comparator are two interfaces that can be implemented so that collections and classes can be sorted.

- public interface Comparable: Defines the compareTo(Object). This method allows one object to be compared to another. The result is referred to as the natural ordering of the objects. Arrays of Objects that implement this interface



INSIGNIA SOLUTIONS

UNTIL NOW, JAVA FOR EMBEDDED SYSTEMS HAS BEEN JUST TOO BIG AND UNPREDICTABLE.

Jeode™

You want the power of Java™. But for your embedded application, it's just too hefty, unpredictable, and not fast enough. Insignia Solutions® introduces Jeode™, our implementation of Java, which is fully compliant with Sun's EmbeddedJava™ specification and uniquely designed for embedded systems. Jeode lets you optimize size, performance and predictability for your specific embedded application. Jeode features our proprietary EVM™ (Embedded Virtual Machine™) that is highly configurable and tunable to achieve optimal performance in resource-constrained embedded systems. The EVM provides adaptive optimizing dynamic compilation for fast execution and concurrent garbage collection for predictable behavior. With more than a decade of delivering virtual machines, Insignia Solutions is already providing Jeode to Fortune 500 makers of networking equipment, PDAs, mass storage systems, consumer electronic devices and more. So, can Java be small enough, fast enough and predictable enough for your embedded system? The answer is a resounding yes! With Jeode. Call **1.800.848.7677** or visit **www.insignia.com** for more.

EVM
EMBEDDED
VIRTUAL MACHINE

Code JR

Insignia Solutions is a registered trademark and Jeode, EVM, and Embedded Virtual Machine are trademarks of Insignia Solutions, Inc. Sun, Java, and EmbeddedJava are trademarks or registered trademarks of Sun Microsystems, Inc. All other trademarks are the property of their respective holders.

can be sorted automatically by List.sort. The interface assumes that concrete implementations of this interface will implement the int compareTo(Object o) method such that the method returns negative if the receiver is less than the object passed to the method, zero if they are equal, and positive if the receiver is greater.

- public interface Comparator: Defines a comparison function which imposes a total ordering on some collection of Objects. Comparators can be passed to a sort method (e.g., Arrays.sort) to allow precise control over the sort order. The Comparator interface specifies one method, the static int compare(Object, Object) method. This method assumes that the result returned is negative if the first object is less than the second, zero if they are equal, and a positive number if the first is greater than the second.

ABSTRACT IMPLEMENTATIONS

A set of abstract classes are provided which implement the Collections API interfaces. These classes are AbstractCollection, AbstractSet, AbstractList, AbstractSequentialList, and AbstractMap. They provide basic implementations for many of the methods specified in their associated interfaces (note the naming convention). The documentation provided with each class in the JDK 1.2 release indicates which methods need to be implemented for a concrete class to be implemented.

AbstractCollection

The AbstractCollection class provides a skeleton implementation of a Collection. This implementation represents what is often referred to as a bag or multiset. Abstractly, a Bag can be considered to be any collection of objects, which can be of any class; these objects are the elements of the Bag. It is a general placeholder for collections of objects. There is no order assumed. It is the most general form of collection available in Java.

To create a concrete implementation of a Collection from the AbstractCollection class it is necessary to implement the following methods:

- public Iterator iterator(): Returns an Iterator over the elements contained in this Collection. This object must implement the hasNext and next methods.
- public int size(): Returns the number of elements in this Collection.

In addition, if the collection is to be modifiable, the subclass must implement the add(Object) method. This is because in the AbstractCollection class, this method throws an UnsupportedOperationException. Note that the methods equals(Object) and hashCode() are inherited from Object.

AbstractSet

This abstract class is a direct subclass of AbstractCollection. It implements the Set interface. The only methods that are defined in this class are the methods equals(Object) and hashCode(). The equals(Object) method checks to see that the two objects are sets, that they have the same size, and that all the objects in the set are equal. It is assumed that the addition methods will conform to the Set constraint that duplicates are not allowed.

AbstractList

This abstract class provides a skeleton implementation of the List interface. Many methods are inherited from its direct super class AbstractCollection. However, it defines methods such as add(int, Object), indexOf(Object), and indexOf(Object, int) from the interface definition so that the basic structure of a List (orderedCollection) is provided. It uses an array to hold the data internally. The data structure used is optimized for random access. If sequential access is to be the most commonly used form of access, then the AbstractSequentialList class should be subclassed.

To create a concrete List collection class, a subclass must implement the following:

- get(int): This returns the object held at the specified position.
- size(): This returns the number of elements held in the collection.

In addition, if the List collection is modifiable the subclass must implement the set(int, Object) method. The default implement will throw the UnsupportedOperationException. The assumption is also made that the List collection is of fixed size. This means that if the subclass is to be "growable" (i.e., can grow in size if necessary), then the subclass must also override the add(int, Object) and remove(int) methods. It is not necessary to override the add() method because the add method actually calls the add(int, Object) method using the size() method to generate the position index.

AbstractSequentialList

This is essentially the same as the AbstractList class form that it inherits except that the data structure and its access methods have been optimized for sequential access rather than random access. This means that it implements the "random access" methods (get(int index), set(int index, Object element), set(int index, Object element), add(int index, Object element), and remove(int index)) on top of List's listIterator, instead of the other way around.

To create a concrete subclass of this class, that subclass must implement:

- public abstract ListIterator listIterator(int index): Returns a ListIterator of the elements in this List (in proper sequence).
- public abstract int size(): Returns the number of elements in this List.

For an unmodifiable List, implement the listIterator's hasNext, next, hasPrevious, previous, and index methods. For a modifiable List, the programmer should also implement the listIterator's set method. For a variable-size list, the programmer should additionally implement the listIterator's remove and add methods. Remember that the get, set, and remove methods all use the lists listInterator.

AbstractMap

This abstract class provides a basic implementation for the Map interface. To create a concrete implementation of a map a subclass must implement:

- public abstract Collection entries(): This method is expected to return a collection of all the key-to-value pairs in the map. It is expected to reflect the current state of the map, thus any changes made to the map should be reflected in the collection. This collection is not expected to support the add or remove methods and its Iterator should not support the remove method.

In addition, if a modifiable Map is to be defined, then the concrete subclass should override this class's put method (which otherwise throws an UnsupportedOperationException), and the Iterator returned by entries().iterator() must also implement its remove method.

CONCRETE IMPLEMENTATIONS

These classes are general-purpose implementations of the core collection interfaces. They all inherit from appropriate Abstract classes and thus provide good examples of how to create concrete versions of the various concepts such as lists and sets (see Table 1). The new classes (i.e., those other than Hashtable and Vector) are unsynchronized, as this greatly improves their performance. They can be synchronized by using a synchronizing wrapper, which can be obtained using the appropriate Collection class static method. For example, to generate a synchronized version of a collection class you can use the static method Collection.synchronizedCollection(Collection c).

HashSet

This class is a direct subclass of the AbstractSet class. It implements the Set interface as well as the java.lang.Clonable and java.io.Serializable interfaces. The HashSet class provides the following constructors:

- HashSet(): New, empty HashSet; the backing HashMap has default capacity and load factor. The default capacity and load factor for a HashMap are 101% and 75%. HashSet(Collection) constructs a new HashSet containing the elements in the specified Collection.
- HashSet(int): New, empty HashSet; the backing HashMap has the specified initial capacity and default load factor (75%).
- HashSet(int, float): New, empty HashSet; the backing HashMap has the specified initial capacity and the specified load factor.

For example, consider the following simple Test class which creates a simple set and adds four strings to it (one of which is a duplicate):

```
import java.util.*;

public class Test {
    public static void main(String args []) {
        HashSet set = new HashSet(10);
        set.add("John");
        set.add("Denise");
        set.add("Phoebe");
        set.add("John");
        Iterator it = set.iterator();
        while (it.hasNext()) {
            System.out.println(it.next());
        }
    }
}
```

An iterator is used to process all the elements in the set. The result of executing this class is:

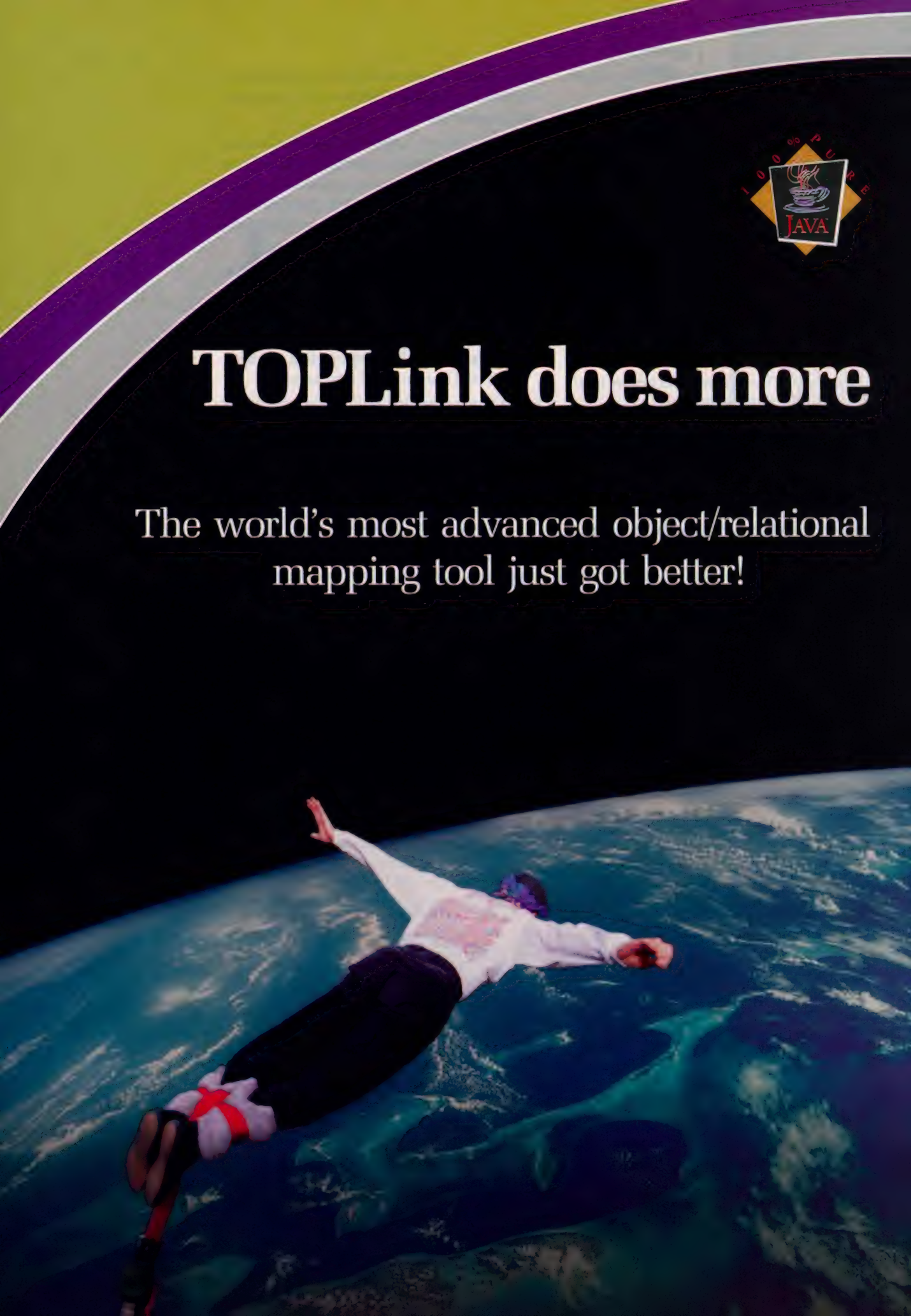
```
C:>java Test
John
Phoebe
Denise
```

ArraySet

This class is a direct subclass of the AbstractSet class. It implements the Set interface as well as the java.lang.Clonable and java.io.Serializable interfaces. It is intended for use with small sets that are likely to be frequently modified. It will give *extremely* poor performance for large sets.

ArrayList

This is a concrete subclass of the AbstractList class. It is a resizable list collection. Essentially, this is an unsynchronized Vector. Like Vector it has an initial capacity and an increment. Like Vector, its default capacity is 10 and its default increment is zero. If the increment is zero, then the size of the ArrayList doubles each time the capacity is exceeded. Like Vector, it possesses



TOPLink does more

The world's most advanced object/relational mapping tool just got better!

TOPLink™
THE OBJECT PEOPLE

Now available from The Object People

TOPLink for Java v2.0

TOPLink allows you to map complex business objects to relational databases so you can leverage your investment in data *and* deploy new applications with Java. **TOPLink** incorporates a meta-data architecture that separates the details of persistence from the business logic. No SQL programming is required.

TOPLink for Java comes with these great features:

- Visual Builder tool for creating mappings
- Flexibility with more than 10 types of mappings
- Object-level transactions
- Queries in terms of objects, not SQL
- Object caching and identity
- Mapping objects to multiple tables, and vice-versa
- Support for any JDBC-compliant database, including DB2, Oracle, Sybase, Informix, SQL Server, and others (also new JDBC 2.0 support)
- Also available - mainframe data extensions

Plus TOPLink for Java v2.0 brings you these new features:

- Java 2 support for collections and weak referenced caches
- Expanded 3-tier support to provide remote unit-of-work functionality
- Pessimistic locking
- Batch writing
- Support for Oracle8i

And TOPLink for BEA WebLogic gives you added EJB support, including:

- Transparent mapping of EJBs to relational databases
- Bean and/or container-managed Entity bean persistence
- Fully integrated functionality with Session beans
- JTS integration

Contact us to enquire about other EJB server support.

For a **FREE** 30-day evaluation copy of TOPLink for Java, or more information, contact us by e-mail at info@objectpeople.com, phone (613) 225-8812, or visit our web site.

www.objectpeople.com

CANADA • U.S.A. • U.K. • GERMANY

TOPLink is a registered trademark of The Object People, Inc Java and Java-based trademarks and logos are registered trademarks of Sun Microsystems, Inc. Other products referred to may be trademarks of their respective owners.

THE OBJECT PEOPLE
www.objectpeople.com

Table 1. Concrete collection classes.

HashSet	Implements a Set
ArraySet	Intended for small sets (faster creation and iteration than HashSet).
ArrayList	An unsynchronized Vector.
LinkedList	A doubly-linked List that may provide better performance if insertion and deletion occur frequently.
Vector	As in JDK 1.1. Included for legacy applications.
HashMap	An unsynchronized Hashtable.
ArrayMap	Intended for small hashtables (faster creation and iteration than HashMap).
TreeMap	A balanced binary tree implementation of the Map interface. Unlike Hashtable, imposes an ordering on its elements.
Hashtable	As in JDK 1.1. Included for legacy applications.

methods to manipulate its size such as ensureCapacity(int), trimToSize(), and size(). The ArrayList class provides the following constructors:

- ArrayList(): An array list with a capacity of 10 and an increment of zero.
- ArrayList(Collection): An ArrayList containing the elements of the specified Collection, in the order they are returned by the Collection's iterator. The increment is defaulted to zero.
- ArrayList(int): An empty ArrayList with the specified initial capacity and the default increment.
- ArrayList(int, int): An empty ArrayList with the specified initial capacity and capacity increment.

Consider the following Test class modified to use an ArrayList. This class creates a simple set and adds four strings to it (one of which is a duplicate):

```
import java.util.*;

public class Test {
    public static void main(String args []) {
        ArrayList list = new ArrayList(10);
        list.add("John");
        list.add("Denise");
        list.add("Phoebe");
        list.add("John");
        Iterator it = list.iterator();
        while (it.hasNext()) {
            System.out.println(it.next());
        }
    }
}
```

The results obtained are:

```
C:>java Test
John
Phoebe
Denise
John
```

Note that ArrayList allows duplicates.

LinkedList

The LinkedList class is a concrete subclass of the AbstractSequentialList class. It provides a doubly-linked list implementation of the List interface. This class is particularly good for sequential access-oriented operations and for insertions and deletions. It provides the following constructors:

- LinkedList(): An empty LinkedList.
- LinkedList(Collection): A LinkedList containing the elements of the specified Collection, in the order they are returned by the Collection's iterator.

The LinkedList class also provides the following additional methods for adding objects to the front and back of a list:

- addFirst(Object): Inserts the given element at the beginning of this List.
- addLast(Object): Appends the given element to the end of this List.

In turn, it also provides the following additional access and deletion methods:

- getFirst() and getLast(): Return the first (or last) Element in this List.
- removeFirst(): Removes and returns the first Element from this List.
- removeLast(): Removes and returns the last Element from this List.

HashMap

This class is a concrete subclass of the AbstractMap class. It is an unsynchronized Hashtable. Like, the hashtable class, it possesses a default capacity and loading factor. In HashMap, these are 101% and 75%, respectively. The constructors provided by HashMap are:

- HashMap(): A new, empty HashMap with a default capacity of 101% and load factor of 75%.
- HashMap(int): A new, empty HashMap with the specified initial capacity and default load factor.
- HashMap(int, float): A new, empty HashMap with the specified initial capacity and the specified load factor.
- HashMap(Map): A new HashMap with the same mappings as the given Map.

An example of using a HashMap is provided below:

```
import java.util.*;

public class MapTest {
    public static void main(String args []) {
        HashMap table = new HashMap(10);
        table.put("John", "C47");
        table.put("Denise", "D57");
        table.put("Phoebe", "A12");
        System.out.println("Johns office is " +
```

```
table.get("John"));
        }
    }
```

The result of executing this class is:

```
c:> java MapTest
Johns office is C47
```

ArrayMap

This is a concrete subclass of AbstractMap. It is intended for small but fast hash tables. It is implemented on top of an ArrayList and overrides the entries(), clone(), and put(key, value) methods. In an ArrayMap the key and value pairs are guaranteed to be in the order in which they were inserted. If a key-value pair is removed and a mapping for the same key is inserted at a later time, the mapping goes to the end of the map.

TreeMap

This class is a concrete subclass of the AbstractMap class and implements a binary tree as the data structure for holding the mappings from keys to values. The binary tree is ordered on the keys and thus guarantees that the key-value pairs are in key ordered. This class thus provides guaranteed log(n) time cost for the containsKey, get, put, and remove operations.

THE Collection CLASS

The Collection class is a general utility class of the Collections API. It provides many very useful (static) methods that can be used to wrap collections in a synchronized wrapping, to search and sort collections, to find the minimum or maximum value in a collection, and create unmodifiable versions of modifiable collections. This class provides no instance-side methods—all the methods are static. The following list summaries the set of methods available:

- binarySearch(List, Object) and binarySearch(List, Object, Comparator): Perform a binary search on the list for the specified object (optionally using the specified comparator).
- enumeration(Collection): Returns an enumeration object for the specified collection.
- max(Collection) and max(Collection, Comparator): Return the object which returns the maximum value of the collection. Either the default comparator for the element is used or a specified comparator must be provided.
- min(Collection) and min(Collection, Comparator): Return the object which returns the minimum value for either the normal comparison method or the specified comparator.

Jumping Beans

Fully exploit the capabilities of Java with this intuitive and powerful mobility framework allowing never-before possible solutions. With the Jumping Beans framework, a developer can quickly create applications which dynamically jump from computer to computer during execution, in a secure, robust fashion.

Jumping Beans enables many new solutions in:

- Network management
- Device management
- Workflow
- Database access
- Business process implementation
- Data gathering
- much more

Full-featured evaluation copy at:
www.JumpingBeans.com
or call 408-738-4616

"Jumping Beans" and the Bouncing Bean Logo are trademarks of Ad Astra Engineering, Inc.
Java, 100% Pure Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

- nCopies(int, Object): Generates a list containing the specified number of copies of the object. This list is immutable.
- sort(List) and sort(List, Comparator): Sort the specified List into ascending order, according to either the natural comparison method or the specified comparator.
- subList(List, int, int): Generates a sublist consisting of the elements of the specified list between the first and second indexes. The first index is inclusive, the second index is exclusive.

The Collections class also provides a set of methods for wrapping a collection within a synchronized wrapper. These methods all have the format synchronized<type of collection>. For example, synchronizedList(List) (which returns the List wrapped in a synchronization wrapper) and synchronizedMap(Map) (which returns the Map wrapped in a synchronization wrapper).

There are also methods for generating unmodifiable collections such as unmodifiableList(List) and unmodifiableSet(Set).

ITERATION OVER COLLECTIONS

Just as with the classes Vector and hashtable, it is possible to iterate over the contents of any of the collection classes. In JDK 1.1, implementations of the Enumeration interface were used to perform this iteration. In JDK 1.2, iterators are used. The change in name is intended to indicate that iterators are more powerful and are meaningful (in that a number of the methods have changed their names to be more obvious). For example, the hasMoreElements() method of Enumeration is replaced by hasNext() as the method next() is used to access the next element in the iteration. It is also possible to modify the collection underlying the iteration while the iteration is in progress. It is expected that the Iterator will replace the Enumeration.

The Iterator interface is one of two interfaces (the ListIterator is the other) which provide for iteration. This interface defines the following (abstract) methods that are implemented by any concrete iterator:

- hasNext(): Returns true if there is at least one more element to process.
- next(): Returns the next element in the collection to process. To access the first element in the iteration, the method next() must first be called.
- remove(): Deletes the last element returned by the method next() from the underlying collection.

The Iterator interface has a direct sub interface, the ListIterator interface. This interface extends the concept of iterators to allow much greater manipulation of the underlying collection as well as the ability to process elements of the iteration in either direction (i.e., backward as well as forward).

The ListIterator interface defines the following (abstract) methods (in addition to next() and remove() defined in the Interator interface) that are implemented by any concrete ListIterator:

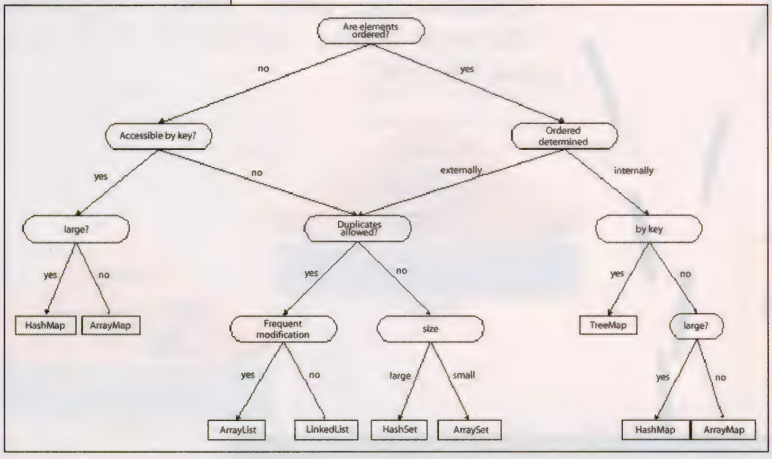
- add(Object): Adds the Object to the underlying list. The object is added at the current point in the iteration. Thus, a subsequent call to next() would return this object.
- hasNext() and hasPrevious(): Return true if there is at least one more element to return using the next() or previous() method.
- nextIndex(): Returns the index of the element that would be returned by a subsequent call to next.
- previous(): Returns the element before the current element in the iteration.
- previousIndex(): Returns the index of the element that would be returned by a subsequent call to previous.
- set(Object): Replaces the last element returned by next or previous with the specified element.

ARRAY SORTING AND SEARCHING

The Arrays class provides a set of utility methods for sorting and searching arrays as well as for converting arrays to lists. There are a wide range of sorting and searching methods for different basic data types (e.g., byte, short, long, etc.) as well as for objects (reference types). The search methods are all of the format binarySearch(<array of type>, <data of type>), for example, binarySearch(byte[], byte). In turn, the sorting methods are all of the format sort(<array of type>), for example sort(short[]). For arrays of type Object, an additional sorting method is provided in which the type of comparator to be used can be specified (sort(Object [], comparator)). The toList method takes an array of type Object and returns a list.

Figure 3.

Selecting which Collection class to use.



Fast, Easy Application Development. . .

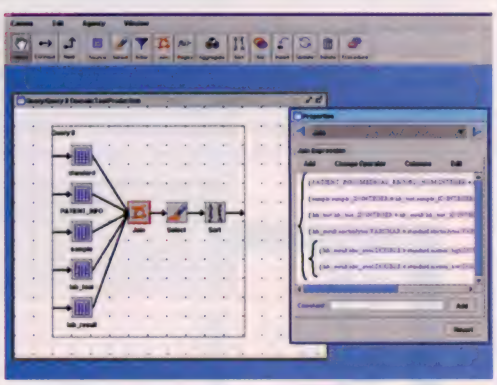
Building something hasn't been this intuitive since kindergarten.

Creating with blocks was fast. It was easy. It didn't require extensive training and planning. You could build whatever you wanted. And you could rearrange the blocks at any time.

Creating new applications can be just like playing with blocks. Cerebellum™ is a data-independent, scalable platform written in Java™ that removes the challenges and problems associated with application development. Cerebellum's single, drag-and-drop graphical user interface accesses, integrates, and manages data from any source in any location. With just a few clicks, you develop data access for new applications. No SQL needed. And with Cerebellum's GUI, queries can be deployed across different environments without any redesign — they are simply bound to new data sources. It's as easy as switching blocks.

With Cerebellum, your focus isn't on building with code; you're building with your mind.

(Your kindergarten teacher would be so proud.)



Try Cerebellum's building-block technology for free. To download, visit

www.cerebellumsoft.com

Toll free: (888) 862-9898

Cerebellum SOFTWARE

Declare Data Independence

Cerebellum is a trademark of Cerebellum Software, Inc.
Java is a trademark of Sun Microsystems, Inc.

Too rushed to optimize?

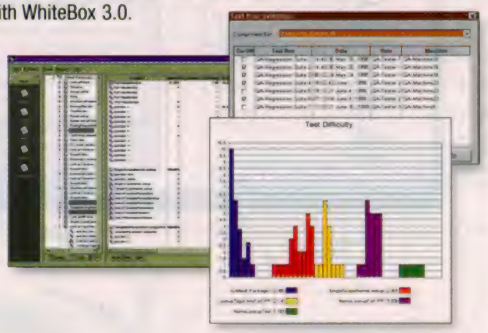
Think again...Introducing WhiteBox 3.0 test optimization and coverage all-in-one.

Getting your software out the door on time can put pressure on quality; that can mean shipping late or continually going over budget.

Test optimization is the solution. With WhiteBox 3.0, a complete code analysis and test optimization tool consisting of metrics and coverage. WhiteBox 3.0 allows developers and QA professionals to determine *where to test, how much to test,* and *when to stop testing,* greatly reducing testing costs. WhiteBox 3.0 optimizes your testing resources and provides metrics to meet your most important project quality needs.

WhiteBox 3.0 is specifically designed to not change the build process. It provides superior scalability, robust analysis with reporting and export capabilities, in addition to the ability to merge results throughout the project lifecycle and multiple test runs. WhiteBox 3.0 supports projects written in **Java**, as well as **C** and **C++**. And if a project is written in all three languages, it provides test optimization and coverage across all the source code. Developers will also appreciate how WhiteBox 3.0 conveniently integrates with Microsoft DevStudio and Symantec's Visual Cafe.

So don't let the time-to-market rush affect either software quality or project cost. Test smarter, test faster, and save money with WhiteBox 3.0.



WhiteBox VERSION 3.0

Reliable Software Technologies

21515 Ridgetop Circle, Suite 250, Sterling, VA 20166
703-404-9293 • Fax: 703-404-9295 • **whitebox@rstcorp.com**

www.rstcorp.com/whitebox
1-800-824-0022

A brief example of the sorting method is provided:

```
import java.util.*;

public class SortTest {
    public static void main(String args []) {
        for (int I =0, I < args.length; I++)
            System.out.println(args[I]);
        Arrays.sort(args);
        for (int I =0, I < args.length; I++)
            System.out.println(args[I]);

    }
}
```

The result of executing this class is:

```
c:> java SortTest John Paul Phoebe Denise
Denise John Paul Phoebe
```

CHOOSING A COLLECTION CLASS

It can sometimes be confusing for those new to the Collection API to decide which collection class to use. Some make the mistake of always using an array (because it is similar to the constructs that they are used to). However, this is failing to understand the way one should work with collections (data structures) in Java. The following decision tree may be of use.

The most commonly used collection classes are HashSet, ArrayList, HashMap, and LinkedList.

Consider the following scenario: "We are about to implement a piece of code which will read records from a database and hold them in memory. In general, only 20 or 30 records at most will be returned. These records will then be viewed by the user. One of the elements will be selected and further action taken." Which collection class should be used? We need to ask the following questions:

- Are the elements ordered? Yes.
- How is the order determined? Externally (i.e., via the order in which they are returned from the database).
- Are duplicates allowed? No (this would imply something bizarre in terms of the same record being returned by the database twice for the same search!).
- What size is the collection? Small.

This takes us directly to the ArraySet class.

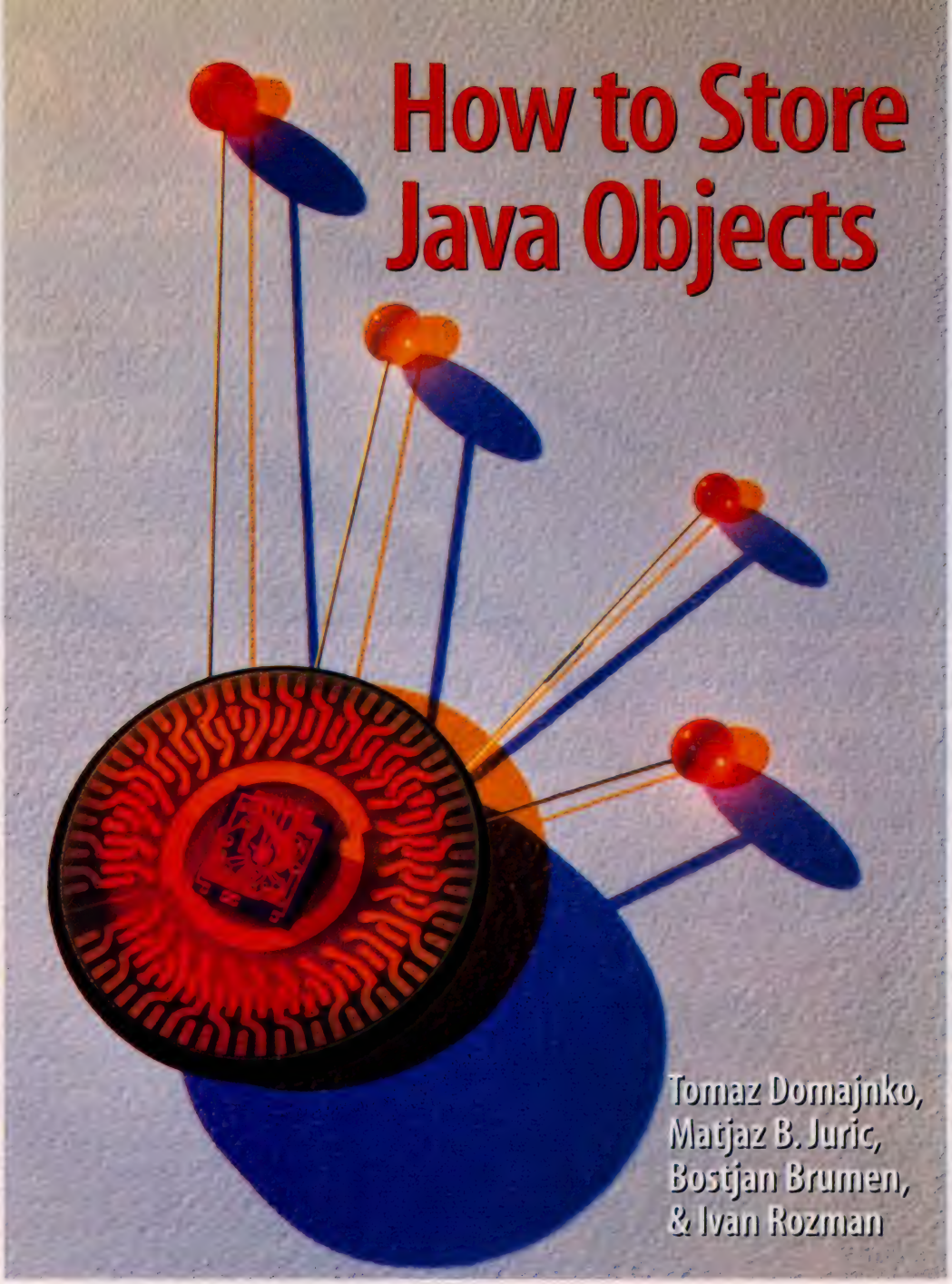
SUMMARY

The Collection API interfaces and classes will form the basis of the data structures you build and will be the cornerstone of your implementations. If you are coming from a Lisp style language, these concepts won't seem too strange. However, if you come from languages such as C, Pascal, or Ada, you may well have found the idea of a bag and a set quite bizarre. Try them out, implement some simple programs using them and you will find that they are easy to use and extremely useful. You will come to wonder why every language doesn't have the same facilities! ■

Dr. John Hunt has extensive object-oriented experience in industry and academia. He can be contacted at john.hunt@jttc.demon.co.uk.

Quantity reprints of this article can be purchased by phone: 717.560.2001, ext.39 or by email: sales@rmsreprints.com.

How to Store Java Objects



Pierre Yves-Goavec/Image Bank

Tomaz Domajnko,
Matjaz B. Juric,
Bostjan Brumen,
& Ivan Rozman

The right choice of a database system is a strategic business decision that has a profound effect on the near- and long-term success of any project

JAVA IS BECOMING increasingly important in mission-critical system development. Advantages accruing from the development process compared to conventional programming languages are well known—Java is platform independent, allowing simple multi-platform system development with the opportunity to provide efficient and powerful data persistence.

Implementing a robust persistent architecture is one of the most challenging tasks in designing a component-based application. Not only does persistence have a pervasive impact on the overall system structure, it also has a significant effect on development schedules and runtime performance of the resulting applications. Implementing persistence in object-oriented programs is especially challenging, because object-oriented programs typically consist of a large number of complex objects.

During development of the Hotel Information System, we were faced with the problem of choosing an appropriate database management system (DBMS). We had to evaluate various database management systems from the perspective of storing complex data. At the time, we had little reliable knowledge for choosing appropriate DBMS for storing objects. We had to devise an approach that would point out the most significant factors for choosing a suitable database.

Performance evaluation is an important component in the process of selecting an appropriate database management system for a project. It is very difficult to compare different database systems because the optimization of the tasks depends on understanding the requirements of the application as well as the DBMS's recommended sequence of proprietary API procedures. Factors that affect application performance range from the data model design and code construction, all the way to the frequency and the size of the commits. These aspects contribute to the complexity of the determining process for choosing an appropriate DBMS for a particular application.

We have found that it is not possible to judge the viability of a DBMS on one factor alone and that several factors proved to be very important when selecting a database system. We have found a firm set of factors important for the final decision when selecting a DBMS.

We describe implementations of persistent Java objects in various database management systems (Oracle 7.2 RDBMS, Poet 5.0 ODBMS, and ObjectStore 2.0 ODBMS). Because technical requirements and constraints are highly domain- and environment-dependent, there is clearly no single persistence strategy that would work equally well for all applications.

Listing 1.

Declaration for class Reservation.

```
public class Reservation {
    // attributes of class Reservation
    private Date startDate;
    private Date endDate;
    ...
    // associations to other objects
    private Guest guest;  // association to guest who created reservation
    private Room room;    // association to the room, reservation points at
    private Vector invoices; // associations to an array of invoices
    // constructor for the class - creates new reservation on parameters
    public Reservation(Guest g, Room r, Date arrival, Date departure, ...) {
        guest=g;  // connect to the guest
        room=r;   // connect to the room
        // set the other attributes
        startDate=arrival;
        endDate=departure;
        ...
        invoices = new Vector(); // array of invoice references creation
    };
    // method that creates another invoice for the reservation
    public Invoice createInvoice() {
        // when creating invoice, we have to get the price from the room...
        float sum=room.getPrice()*calcDuration();
        Invoice invoice=new Invoice(...);
        invoices.addElement(invoice);
        ...
    };
    public float calcDuration(){...};
    // methods that support navigation
    public Invoice getInvoice() {...};
    public Room getRoom() {...};
    public Guest getGuest() {...};
    ...
};
```

RELATIONAL AND OBJECT DATABASES COMPARISON

Object database systems offer native support for storing objects. The objects can be stored after performing two phases. First, they have to be made persistent by adding functionality for object storing. It can be done by using database application programming interface (horizontal approach) or by inheritance (vertical approach). When using the horizontal approach the class declarations do not have to be changed, but changes to the code that stores/retrieves the objects are necessary. If the vertical approach is used, the class declarations have to undergo minor changes—they have to inherit from a persistent-capable base class of the database system. Object database systems simplify the task of making objects persistent by introducing a preprocessor, which automatically prepares objects for storing in an object database (the preprocessor uses either a minimal extension to the Java language or a configuration file to mark classes as persistent-capable). The second phase is application specific—the application has to instantiate objects and explicitly store them into the database. This task is usually accomplished by a call to a single method. Here, we have used object database systems that natively support Java. For this reason, many changes have not been made to the basic object model.

JAVA ACCESS TO RELATIONAL DATABASES

When connecting Java applications to relational database systems a set of Java Database Connectivity (JDBC) classes can be used. The classes were introduced by JavaSoft and represent a standard SQL database interface.

JDBC is built in several layers. Application interface layers consist of a limited number of classes (`Connection`, `Statement`, `PreparedStatement`, `CallableStatement`, and `ResultSet`) in the `java.sql` package.

Database interface layers can be implemented in four different ways. First, the *JDBC–ODBC Bridge* can be used. This approach uses proven technology of Open Database Connectivity (ODBC) and only links JDBC and ODBC layers. It enables Java applications to access a wide range of database systems with ODBC support, but normally database client code is loaded on each client machine.

Second, a *native-API partly-Java driver* can be used which converts JDBC calls into calls on the client API for different DBMSs. Like the bridge driver, this kind of driver requires that some binary code is loaded on each client machine.

Third, a *net-protocol all-Java driver* translates

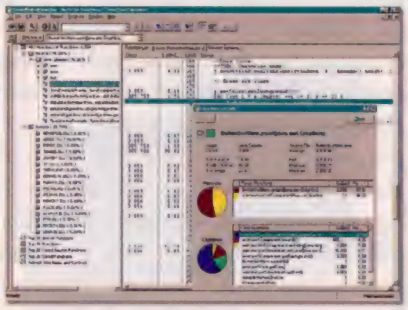
Are your Java tools trapped in ONE virtual machine?

Unleash your Java™ development with DevPartner™ for Java

If you're like most Java developers, your tools are trapped in a *specially modified* virtual machine (VM), with different runtime characteristics than production VMs your code will be deployed on. If your Java code has to perform and run reliably on more than one VM, you'll appreciate the multi-VM power of NuMega DevPartner for Java. DevPartner for Java works with the same *unmodified* VMs that your code will run on in the real world, which can dramatically reduce your debugging, tuning and testing downtime.

DevPartner for Java includes:

JCheck
Visual Thread and Event Analyzer



TrueTime Java Edition
Automatic Performance Analysis and Optimization

TrueCoverage Java Edition
Automatic Code Coverage Analysis

DevPartner for Java Delivers:

- Multi-thread runtime analysis and error detection.
- Code coverage and stability measurement.
- Complete top-to-bottom profiling.
- Powerful VM-switching on the fly.
- Compatibility with the widest range of Java 2 and JDK 1.1 development environments, including Symantec Visual Cafe, IBM Visual Age for Java, Inprise JBuilder, Microsoft Visual J++ and more.

Call for a FREE 14-day license, or visit NuMega on the web

1-888-686-3436
www.compuware.com/numega

COMPUWARE
People and software for business applications™

30-Day Money Back Guarantee

Copyright © 1999 Compuware Corporation. All rights reserved. All trademarks and registered trademarks are the property of their respective owners.

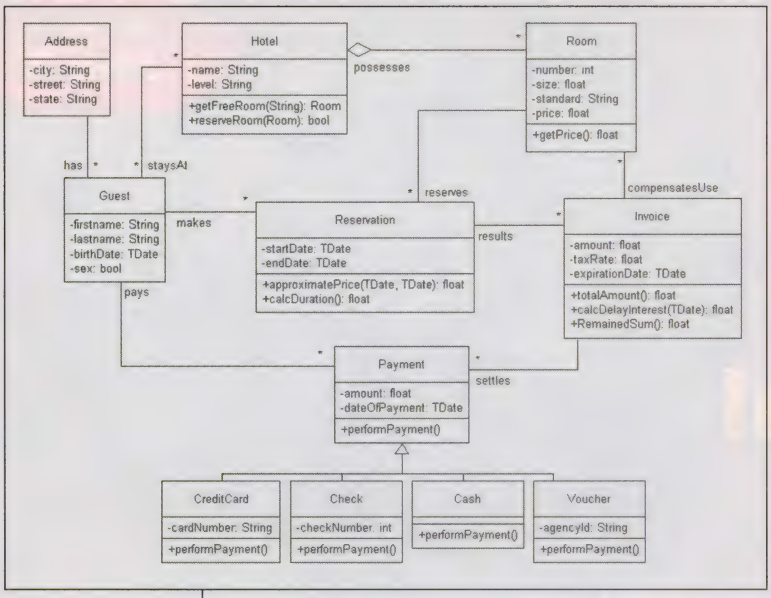


Figure 1.

Hotel Information System object model.

JDBC calls into a set of DBMS-independent commands of net protocol, which are then translated to DBMS commands. This net server, Middleware, is able to connect all its Java clients to many different databases. In general, this is the most flexible JDBC alternative. It is likely that all vendors of this solution will provide products suitable for Intranet use.

Fourth, a *native-protocol all-Java driver* converts JDBC calls into the network protocol commands used by DBMSs directly. This allows a direct call from the client machine to the DBMS server and is a practical solution for Intranet access. Because many of these protocols are proprietary, the database vendors themselves will be the primary source for this kind of driver.

We selected the JDBC–ODBC bridge approach. This is the most frequent method of access to relational databases. With this approach we can use existing ODBC installations and assure co-existence of new systems with legacy applications.

Connecting Java applications to relational database systems requires much more work; enabling access to an RDBMS is only the first. The second step is the mapping of objects to relational schema. This can be accomplished with available commercial tools or with a development of mapping mechanisms between the object model and a relational model. Because our goal was not only to determine performance degradation, but also to evaluate the efforts required to complete the step, we have implemented the mappings ourselves.

Listing 2.

Example of Poet configuration file.

```
[classes\NonPersistentClass]
persistent = false  // transient class, not persistent
                    // capable
[classes\Reservation]
persistent = true  // class made persistent
```

TEST CASE

A part of our approach to evaluate DBMSs was to take a small portion of the Hotel Information System and implement it using different DBMSs. This way the results obtained will be a good measure and a basis for the subsequent implementation of the whole system. A simplified object model is presented in Figure 1. The primary goal of the model is to keep track of hotel guests, room reservations, and bookkeeping activities. Some presumptions were made (e.g., Reservation is used both for Room reservation and for Invoice creation) and some classes that should occur in the model were left out or merged with existing classes. The object model in Figure 1 presents only the most important attributes and methods of classes for understanding the problem, and filters out attributes and methods that are of no significant importance.

Implementation of a test case in Java

Although the objects are simplified for this demonstration, the relationship between the objects have remained complex. The central class for the control of the hotel guests and the payments is the Reservation class. Listing 1 is a sample of the Java code that implements the class Reservation with no support for the database. This represents a basis from which our testing will start.

The central part of the Reservation class is the implementation of the connections to the classes Guest, Room, and Invoice. In case of simple Java implementation, those connections are modeled either as simple object references (connections to Guest and Room) or as Vector (connection to the set of Invoice objects).

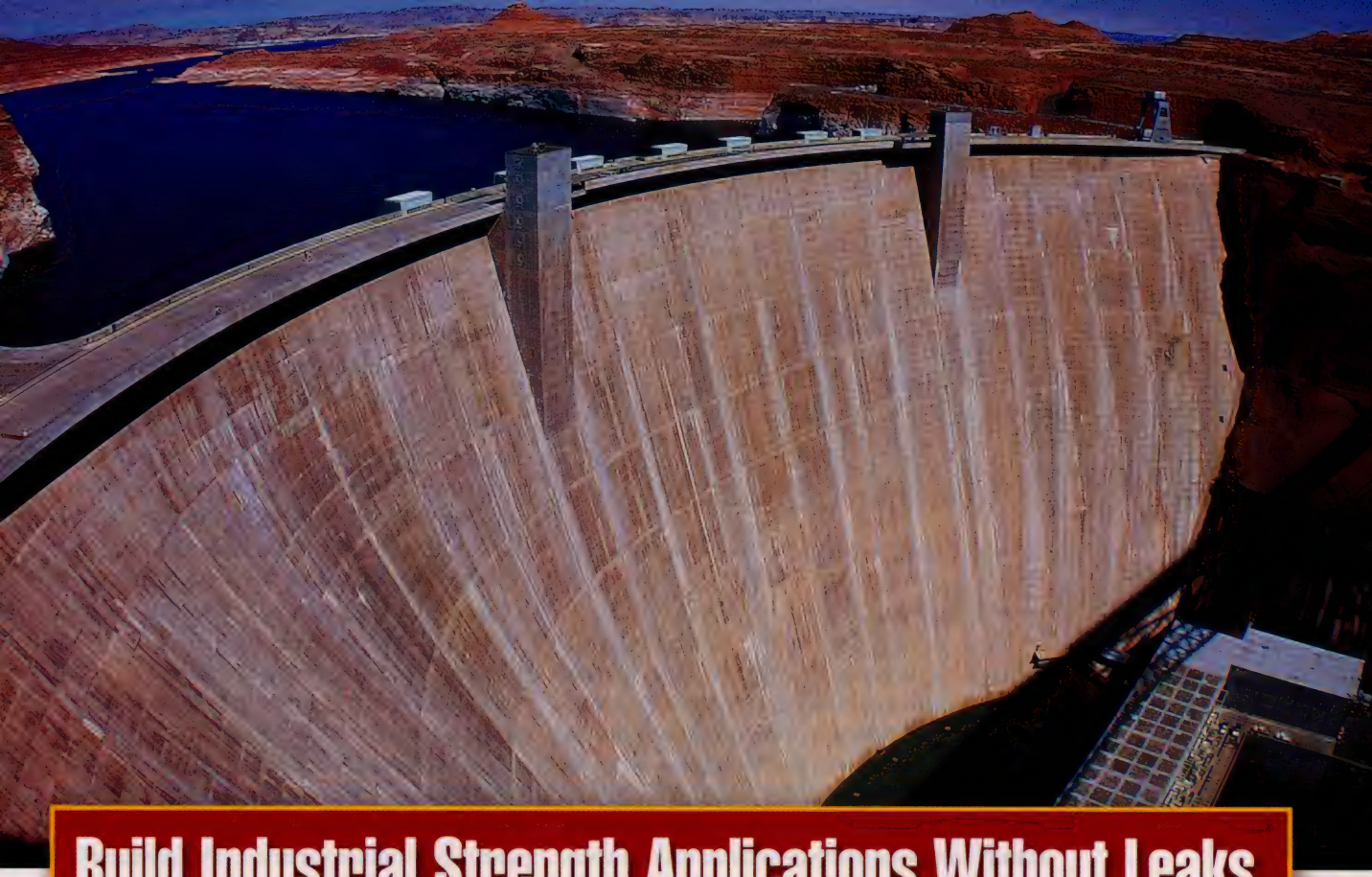
Using an object database

First, we describe the implementation of a partial object model using object databases—Poet and ObjectStore PSE Pro.

Poet

Poet uses a preprocessor to prepare classes to be persistent. To enable persistence, a user first has to create a configuration file where the classes to be persistent are defined. Listing 2 shows an example of a configuration file for a Poet object database where one transient and one persistent class are defined.

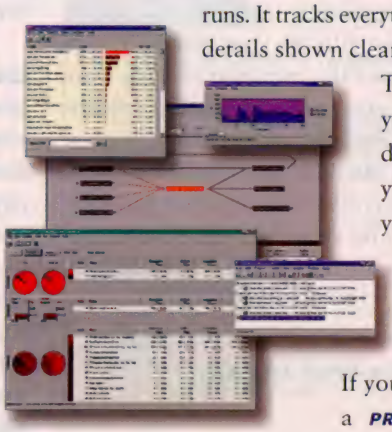
Second, a user has to replace definitions of native Java collection classes with persistent-capa-



Build Industrial Strength Applications Without Leaks

Whether you're building a high performance hydro-electric dam or a ***MISSION-CRITICAL JAVA APPLICATION***, you know that those pesky leaks can sabotage everything! You need a professional tool that is accurate, easy to use and helps you deliver ***SCALABLE, ROBUST PERFORMANCE*** from start to finish.

JPROBE PROFILER is used by Java developers around the world to improve the performance of their Java applications. And now, the most ***POWERFUL TOOL*** for finding and eliminating Java performance bottlenecks makes it easy for you to find and plug memory leaks in your Java code. Use the new ***HEAP REFERENCE GRAPH*** to instantly see a heap snapshot analysis and untangle even the most complex object references that prevent garbage collection.



JPROBE Profiler

JProbe Profiler gives you the most ***ACCURATE MEASUREMENT*** of both time and memory consumption while your program runs. It tracks everything for you, thread by thread, with method details shown clearly right back to each line of source code*.

The advanced ***CALL GRAPH*** helps you analyze your profiling snapshot and quickly drill-down to performance bottlenecks. And you'll get superior performance where you want it because JProbe Profiler provides ***MULTI-PLATFORM*** support for Windows NT and Solaris in JDK1.1, JDK1.2, and JIT environments.

If you develop Java code for a living, you need a ***PROFESSIONAL JAVA PROFILER.*** Get a free evaluation copy of ***JPROBE PROFILER*** right now and start making your Java code industrial strength, without leaks.

Try JProbe Now! www.klgroup.com/jprobe | ***The Easiest Way to Search and Destroy Java Performance Bottlenecks***

KL GROUP
Software Development Productivity

KL Group Inc. Toll-Free: 1-888-361-6205 Tel: 416-594-1026 Fax: 416-594-1919
KL Group Europe B.V. Tel: +31 (0)20 679 95 03 Fax: +31 (0)20 470 03 26
www.klgroup.com

* Source code not required to profile with JProbe. JProbe is a trademark of KL Group Inc. All other products are trademarks or registered trademarks of their respective companies. JR003

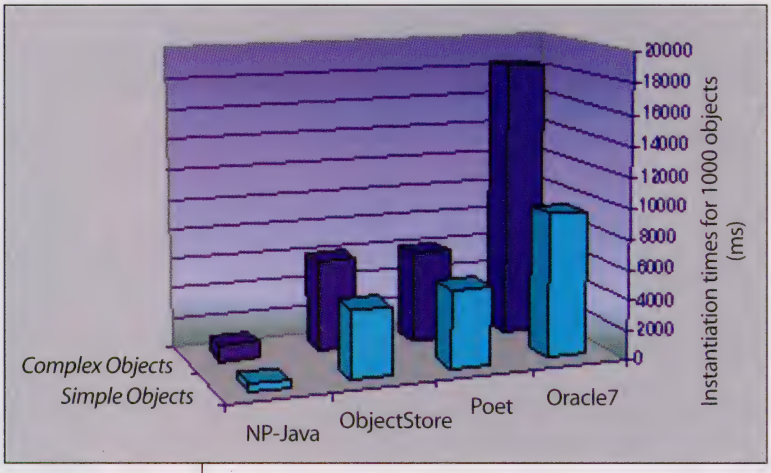


Figure 2.

Instantiation times comparison.

ble collections because native Java collection classes do not have the functionality to support persistent storage of objects in a Poet database.

We have two choices: either we use Poet's collection classes or third-party collections that are persistent-capable. The collection classes that come with Poet implement the same interfaces as original Java collection classes, so we will not have to change any code, just the declaration and instantiation of a collection class. If third-party persistent-capable classes were used, some changes in code would probably be required. We have chosen the first option for two reasons. First, the object model was implemented using Java collection classes (and therefore no change in code is needed) and second, it is highly probable that Poet classes are optimized for use with the Poet database.

Listing 3 shows the required change from "java.util.Vector" class to the persistent capable "COM.POET.odmg.VarrayOfObject" collection class that comes with the Poet database.

With those two minor changes we have enabled objects to be stored in the Poet object database and no further changes are required.

Listing 3.

Declaration of persistent Guest class in Poet object database.

```
public  class Guest{
    // attributes
    ...
    // connections
    private VArrayOfObject reservations;  //(changed from java.util.Vector)
    private VArrayOfObject payments;      //(changed from java.util.Vector)
    private VArrayOfObject hotels;        // (changed from java.util.Vector)
    private Address address;              // (no change, direct object reference)

    public Guest(...) {
        ...
        // changed from creating Vector instances
        reservations= new VArrayOfObject();
        payments= new VArrayOfObject();
        hotels= new VArrayOfObject();
    };
    ...
};
```

We only have to compile the source code with Poet preprocessor.

ObjectStore

Like Poet, ObjectStore also requires two steps for enabling storage of objects. The first step is to annotate the classes that are to become persistent. With ObjectStore, annotations are the additional code needed to make an object persistent. This step is usually completed by running ObjectStore's class file postprocessor on the Java class. The second step is to make sure the annotated classes are located in CLASSPATH before other Java classes. Another issue that must be addressed when using ObjectStore is the handling of built-in classes. Unlike user-supplied classes, built-in classes cannot simply be annotated using the ObjectStore postprocessor, as the Java virtual machine (JVM) does not allow the standard implementations to be replaced with annotated versions for security reasons. For the standard container classes like Vector and Hashtable, one solution is to use "work-alike" versions provided with ObjectStore. For example, java.util.Vector can be replaced by a similar version of the OSVector class from Object Design. An example of the necessary changes is presented in Listing 4. The downside of this technique is that some code modification is required. Another solution to the problem is to use third-party collection libraries instead of the built-in containers.

Using relational database

Making an object persistent using JDBC/RDBMS requires much more development work. The first, and the most difficult, step is to map Java objects to relational tables using available SQL data types. This process could be very simple if we had to implement a class that has no inheritance and includes only simple data types (e.g., strings, integers, etc.). However, if a class includes inheritance and complex Java data types (e.g., hashtables, vectors, etc.) the process becomes quite difficult. In our example, making class Payment persistent requires resolution of inheritance in the relational schema. There are at least two possible solutions to the general mapping issue (to store the entire class in a single table and have a new table for each new class or to develop a hierarchy in the database and store class attributes in more than one table—for each level of inheritance we have a new table), but each requires ease-of-development or performance trade-offs depending on different contexts. For our demonstration we decided to collapse multiple inheritance levels into a single table for each class. The main reason for this design decision

is to minimize the database interaction required. Rather than having to retrieve pieces of information from multiple tables, every persistent class in our system is represented by a single primary database table. The main drawback of this decision is that each class must store all the data members it contains plus all the attributes of parent classes. For example, all the objects displayed in a CreditCardPayment derive from the Payment class. Among other things, the base Payment class provides a private data member amount, which simply stores the amount of payment. Using our table, mapping the amount field must be duplicated in different tables used to store all the classes derived from the Payment class. In some cases, additional tables would be necessary to support complex Java objects. In our example, we have simplified the system in a way that no support was needed for complex objects.

As a result of mapping objects to tables, a many-to-many relation exists in our object model. When mapping a conceptual model to a logical one, many-to-many relations must be resolved by introducing a new connection table (Listing 5). In our case we introduced table "HotelGuest" as a connection between the entities "Guest" and "Hotel."

Once these tables have been created, code must be written to allow a Java class to interact with a relational database. We used relational DBMS Oracle version 7.2 for testing because the system is wide-spread and we have a lot of experience with it.

In our implementation, separate classes were developed containing methods for retrieving, inserting, updating, and deleting objects from the database. We developed a new class for each class in the object model. The new classes contain methods that allow storing/retrieving objects to/from relational database systems. We will not examine those classes in detail, as implementation is straightforward (and time consuming).

MEASUREMENTS AND RESULTS

Object and relational technologies are very distinctive. So we have chosen to test performance on the object model level. Therefore, we had to insure an infrastructure for storing data in various systems and at the same time develop comparable tests on all the systems. It is obvious that this approach can not be as general as desired, but it can be considered as a basis for subsequent tests.

The testing environment was a typical client-server architecture. For clients we used personal computers (Intel Pentium 166 and 200 MMX, 64 MB RAM), connected with a 10 Mbps network to an Intel Pentium-200 server machine with 128 MB RAM. The stage should represent real-world circumstances. We repeated our tests 20 times to get reliable results. Results in the tables are average values.

We designed three groups of performance tests and a complexity measurement. The tests and the measurement should give an indication of important factors to be considered when choosing a database management system. As a reference we measured the times needed for execution of tests without storing objects—we

Listing 4.

Declaration of persistent Reservation class in ObjectStore object database.

```
public  class Reservation {
    // attributes of class Reservation
    ...
    // connections to other objects
    private Guest guest;          // no change needed
    private Room room;            // no change needed
    private OSVector invoices;    // (changed from java.util.Vector)

    // constructor for the class—creates new reservation on parameters
    public Reservation(Guest g, Room r, Date beginDate, Date endDate ...) {
        guest=g;  // connect to the guest
        room=r;   // connect to the room
        // set the other attributes
        startDate=start;
        endDate=stop;
        ...
        invoices = new OSVector();  // OSVector instantiation
        ...
    }
};
```

Listing 5.

Declaration of tables in relational database.

```
CREATE TABLE Hotel (
    _oid NUMBER(19) NOT NULL PRIMARY KEY,
    _name VARCHAR(64) NOT NULL,
    _level VARCHAR(32) NOT NULL
    . . .
);

CREATE TABLE Guest (
    _oid NUMBER(19) NOT NULL PRIMARY KEY,
    _firstname VARCHAR(64) NOT NULL,
    _lastname VARCHAR(64) NOT NULL,
    _birthdate DATE NOT NULL,
    _sex INTEGER NOT NULL
    . . .
);

CREATE TABLE HotelGuest (
    _oidhotel NUMBER(19) NOT NULL,
    _oidguest NUMBER(19) NOT NULL
    . . .
);

CREATE TABLE Reservation (
    _oid NUMBER(19) NOT NULL PRIMARY KEY,
    _startdate DATE NOT NULL,
    _enddate DATE NOT NULL,
    _oidguest NUMBER(19) NOT NULL,
    _oidroom NUMBER(19) NOT NULL,
    _oidinvoice NUMBER(19)
    . . .
);
```

Design
space shuttles
and coordinate
lug nut
inventory.

All from
one outlet.

CO

om

One desktop. No compromises. You need the **power** of a Sun™ workstation while being able to run your PC applications at the same time. That's why we took the muscle behind what makes us the dot in .com and applied it to workstations. The result? The Ultra™ 5 and Ultra™ 10. The SunPCI™ card option lets you run productivity apps and **collaborate** with co-workers, suppliers, the world—while giving you the strength that only Sun can provide. And because they run the Solaris Operating Environment™ software, they are compatible with servers that run up to 64 processors. Plus, they both run on the same robust, **highly scalable** platform that the most powerful Sun workstations do. So, now you can run thousands of business and technical applications without ever altering a thing. All of this at a **price** you'd usually pay for one of those NT workstations (starting at about $2,500). If you're tired of compromise and want more power, call 800-SUN-FIND or visit sun.com/desktop. THE NETWORK IS THE COMPUTER.™

We're the dot in .com.

Sun microsystems

JAVA All pricing is based on MSRP as of 1/99. ©1999 Sun Microsystems, Inc. All rights reserved. Sun, Sun Microsystems, the Sun Logo, Ultra, SunPCI, the Solaris Operating System, the Java Coffee Cup Logo and The Network Is The Computer are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. Ultra 10 shown monitor not included.

Table 1. Instantiation times in milliseconds for a set of 1000 simple objects.

	NP-Java	ObjectStore	Poet	Oracle7
Time	456,5	4503,6	5075	9545,1

Table 2. Instantiation times in milliseconds for a set of 1000 complex objects.

	NP-Java	ObjectStore	Poet	Oracle7
Time	1056,7	6055,2	6060,8	18293,8

Table 3. Navigation times between objects depending on navigation distance.

Distance	1	2	3	4
NP-Java	665,1	1201,1	1246,7	1284,5
Poet	1092,3	1656,1	2097,5	2344
ObjectStore	1125,9	1587,4	2100	2409,8
Oracle7	5754,8	9562,3	14988,2	18150,8

used native Java implementation. These tests are labeled with "NP-Java", meaning non-persistent Java test execution.

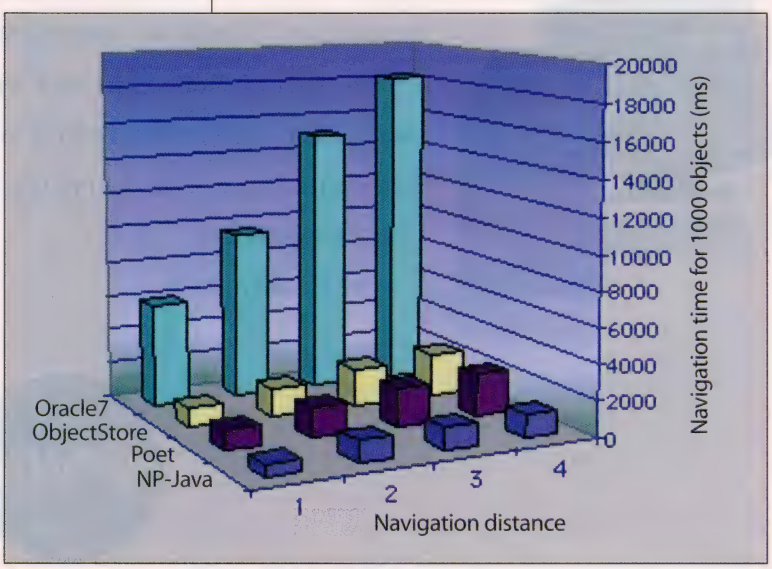
The first group of tests were designed to measure the time needed to instantiate objects. The first test used a set of simple objects (atomic objects not connected into the object graph). We declared them simple, as there is no overhead of connecting them into the graph. All the results are timed in milliseconds and present instantiation of 1000 different objects (we used a combination of objects of type Address, Guest, Reservation, Hotel, and Room). For a comparison of the instantiation times needed to create transient Java objects are added. The results are shown in Table 1.

In the second test we measured how long it takes to instantiate complex objects (connected into the object graph). We classified the objects as complex because besides creating them we also need to connect them into the object graph (as you can see from declaration we have to store the identity of a target object into the source object). To correlate data with the previous measurements we have used the same set of objects, but this time we have established connections between them. The results are shown in Table 2.

The results of these two series of tests are very interesting. As shown in Table 2, the performance decreased when objects were connected into the graph. High-performance degradation when using relational databases can be explained by the need to create objects in link tables—in the case of object databases those link tables do not exist.

The next set of performance tests were designed to measure navigation times in an object graph. We were able to perform these tests because we have in fact implemented an object layer on top of a database system. By having such a layer we were able to measure the speed of the navigation even in the case of a relational database. But the test has gone even further—we have tested the impact of the distance of navigation on the time needed. The distance of navigation was defined as a number of object traversals involved

Figure 3.
Performance degradation from navigation distance.



Put the Leading Experts in C++ Development to Work for You.

Reading *C++ Report* is like attending a specialized, in-depth class in C++ techniques without ever leaving your work station. Every issue examines the most important challenges you face when programming in C++, at the level of detail that you demand.

Regular features include:

- Object-oriented design
- Using design patterns
- Advanced programming
- Getting the most out of STL
- DCOM security
- Writing exception-safe C++
- High performance C++ computing
- Optimizing plain and multithreaded C++ code

Get the publication that is as serious about C++ as you are.

C++REPORT

Subscribe Today

and pay only $79 for 10 issues of C++ Report*

For international prices please see our Web site or call +44 0 1858 435 301

www.creport.com
email: subscriptions@sigs.com
Phone: 800-361-1279 (U.S. only)
615-377-3322 (Canada and Mexico)
+44 0 1858 435 302 (All other countries)

Table 4. Performance degradation from query distance.

Distance	1	2	3	4
Poet	4797,2	5072,4	6599,5	7044,7
ObjectStore	6163,4	6354,7	6633,4	7090,6
Oracle7	3327,8	5154,4	10267,5	18780,5

Table 5. Code size comparison.

LOC	Hotel	Guest	Reservation	Test/Average
NP-Java	182 (100.0%)	143 (100.0%)	280 (100.0%)	415.3 (100.0%)
Poet	183 (100.5%)	144 (100.6%)	281 (100.3%)	482.5 (116.2%)
ObjectStore	212 (116.5%)	163 (114.7%)	308 (110.0%)	454.0 (109.3%)
Oracle7	352 (194.3%)	267 (186.7%)	417 (148.9%)	753.2 (181.4%)

in a transaction [in a relational database the distance n means (n-1) joins to represent data connections]. The times for navigating 100 objects in milliseconds are given in Table 3 and the graphical presentation in Figure 3.

The results show that native navigation support reduces the time needed to access data. Relational DBMSs do not have native support for navigation and therefore have to implement this feature with time-consuming table joins. This explains the rapid growth of measured times. It is interesting to compare the timings between native Java and object database implementation—the overhead of network latency and locking mechanisms can clearly be seen. The overhead is approximately 60%.

The last set of tests were designed to test queries in databases. The tests were classified by their complexity and the number of objects involved. The distance of a query was defined as a number of object types involved in a query (in relational databases it means the number of table joins). The results are given in Table 4 and in Figure 4 (presented are times in milliseconds for performing 100 queries; each query selected a single object from a database with 45,000 stored objects).

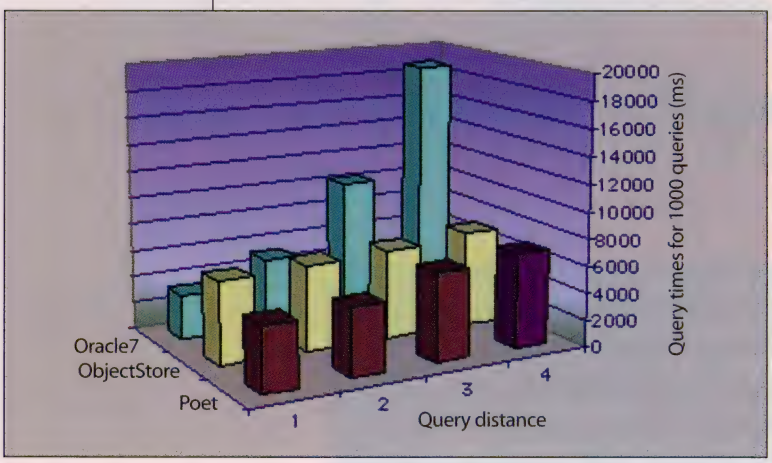
With growing complexity of queries, the time needed for query execution grows, but it grows slowly. Only in the case of the relational database do the query execution times rise rapidly. In our opinion, it is due to the time-consuming table joins. From the result of the test it is obvious that by combining queries and navigational access, better performances can be achieved than by queries without navigational data access.

With the last measurement we tried to find out the effort needed for the implementation of the object model using a specific DBMS. We have measured the LOC* in the implementation classes. The results are presented in Table 5.

The measurement of code size confirmed our hypothesis that by using ODBMS and by storing native objects there will be only a slight increase in development effort. The size of the object model code remained the same and the code size for accessing the database was approximately 10% larger than the code size for native Java implementation (note that the code

Figure 4.

Query time dependency from distance between source and destination object.



* Lines of code, one of the most popular (and misused) software metrics.

Understanding Enterprise JavaBeans™ Technology

AN EDUCATIONAL SEMINAR

Using the Enterprise JavaBeans Component Architecture to Develop Distributed Object Applications

Sun microsystems — THE NETWORK IS THE COMPUTER™

Persistence

Cambridge Management Consulting

SYMANTEC.

Java REPORT

JRUN

April 27	New York
April 28	Boston
April 29	Washington DC
April 30	Atlanta
May 4	Denver
May 5	Los Angeles
May 6	San Francisco
May 7	Seattle
May 14	Research Triangle Park
May 18	Toronto
May 19	Chicago
May 20	Minneapolis
May 21	Houston
May 24	London
May 26	Stockholm
May 28	Paris

Please check our website for additional international dates and locations.

The Enterprise JavaBeans (EJB™) component architecture from Sun Microsystems, Inc. has the potential to dramatically simplify the development of distributed, multi-tier applications incorporating Sun's Java™ technology. Many companies believe that the Enterprise JavaBeans specification may be the most important Java technology offering, second only to the Java programming language itself.

Persistence, Symantec, Cambridge Technology Partners, Live Software and Java Report are among them.

That's why we are joining with Sun Microsystems to offer a day of Enterprise JavaBeans technology education – modeling, developing, deploying, and managing EJB applications.

You will learn how to:

- Use EJB to develop distributed applications
- Use multi-threading, transaction control, state management, and persistent objects
- Use the Java Naming and Directory Interface (JNDI) and the Java Transaction Service (JTS) technologies within an application
- Move directly from a graphical object model to object-oriented data access classes
- Use object-relational mapping and caching technologies for breakthrough performance
- Develop server-side logic and access this logic from distributed applications
- Integrate existing CORBA systems with EJB applications
- Deploy distributed applications to thousands of users
- Reduce the cost and complexity of managing distributed applications and services

THE LEARNING DOESN'T END WHEN THE SEMINAR IS OVER. AS A SPECIAL GIFT, SEMINAR ATTENDEES WILL RECEIVE A COPY OF THE UNDERSTANDING ENTERPRISE JAVABEANS TECHNOLOGY CD-ROM.

Space is Limited: Register Today!

To register, or for more information, please call **1-800-643-3976**, email **info@ejbseminars.com**, or visit our website at **www.ejbseminars.com**.

Mention code 1A to receive 5% off your registration fee.

$149.00 US	Register 2 or more weeks prior to event
$199.00 US	Register less than 2 weeks prior to event
$99.00 US	Price per person if three or more individuals from the same company register simultaneously 2 or more weeks in advance

www.ejbseminars.com

needed for exception handling is not included in the measurement).

In the case of using RDBMSs there is a substantial amount of additional effort needed when compared to native Java implementation. This is due to the fact that no wizards or supplied components were used—we wanted to compare pure program code. In the real world a large number of tools exist in the market, which allow a developer to visually edit the model and to generate required code. Almost every Java development environment for relational databases includes a set of components and wizards and thus simplifies the development.

CONCLUSION

Building state-of-the-art applications often require that cutting-edge technology be used. The selection of a database system on which to build an organization's applications is a complex process in which diverse and sometimes competing sets of business and technology criteria must be satisfied. Making the right choice is ultimately a strategic business decision that has a profound effect on the near- and long-term success of any project.

Using only performance tests can not give the answer to the question "Which database system to use for storing Java objects?" Choosing a database system is not only a very important technological question—it is also a political one. To avoid choosing the wrong database system, we suggest performing some initial analyses of candidate database systems at a high level. Next, a target application should be modeled. It is important that the benchmark models those areas of the target application in which the performance of the database is crucial. After conducting benchmark tests some very important aspects should be addressed: ease-of-use, implementation times, development costs, and maintenance efforts.

Our tests confirmed that only by using object database systems can the full potential of object paradigm be exploited. We exposed performance degradation of relational systems in the case of complex data models. ■

Tomaz Domajnko, Bostjan Brumen, and Matjaz B. Juric are researchers at the University of Maribor in Slovenia; Ivan Rozman is a professor at the University of Maribor. They can be contacted at tomaz.domajnko@uni-mb.si, bostjan.brumen@uni-mb.si, matjaz.juric@uni-mb-si, and i.rozman@uni-mb.si, respectively.

Quantity reprints of this article can be purchased by phone: 717.560.2001, ext.39 or by email: sales@rmsreprints.com.

Java Training for Developers Who Need to Deliver

Petronio Technology Group's industry-acclaimed Java training courses can help accelerate your Java learning curve and keep you abreast of today's latest Java technologies. Advanced topics include JDBC, Beans, I18N, Security, JNI, RMI and Serialization. Popular hands-on Java training workshops include:

- Ultimate Java Programming
- JavaBeans
- Advanced Java Programming
- JFC

* Public Training Coming to a Location Near You. See www.petronio.com/locator for details.

Now offering JavaBeans Java 1.2 & JFC

"Outstanding instruction and customized course materials. The mix of lecture and labs gave us a great jump start in meeting our C++ and Java needs."
Mark Romagna, Regional Technical Manager, Sybase, Inc. - Powersoft Business Group

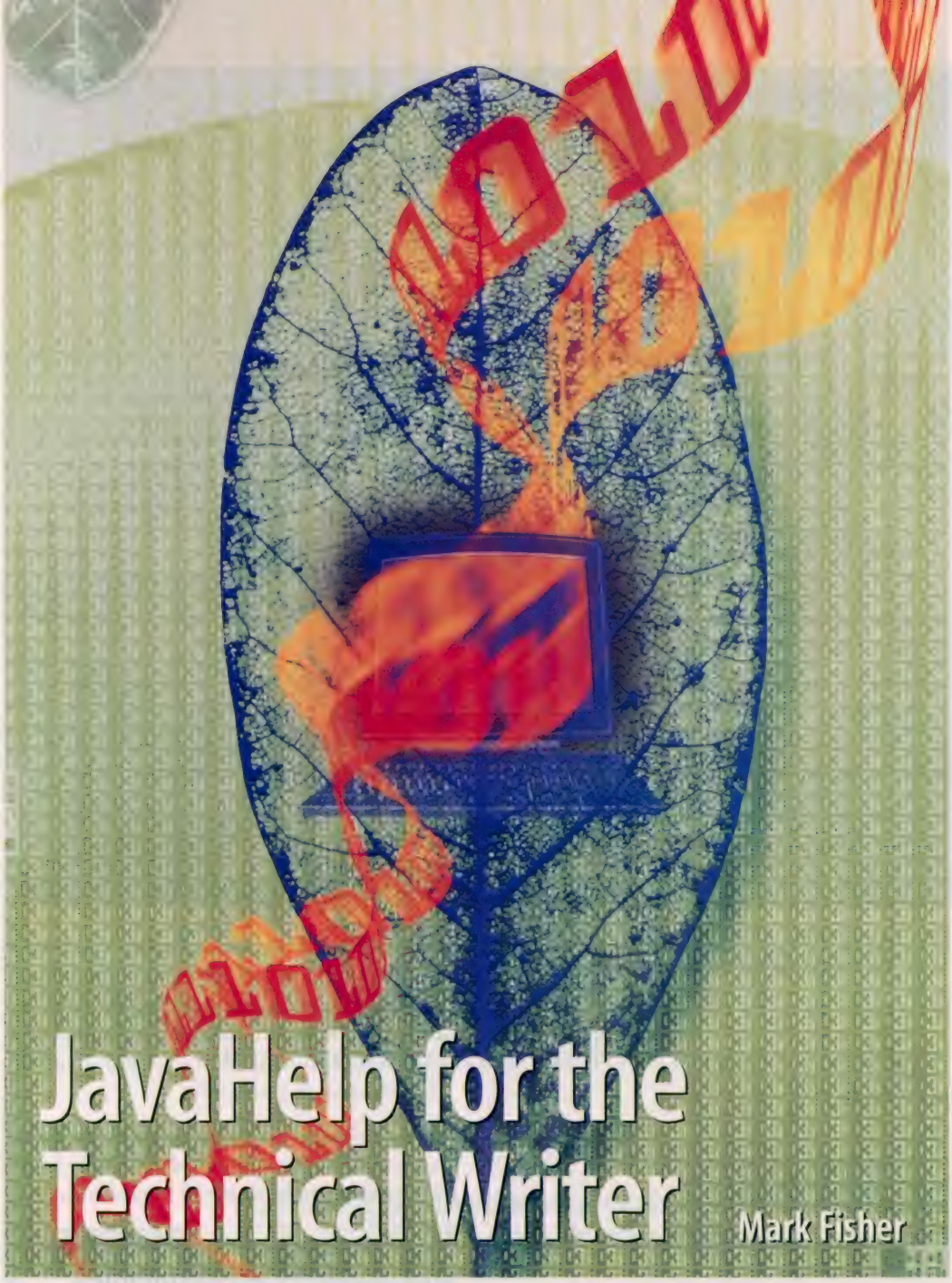
"I highly recommend Petronio Technology Group's Java courses to any professional development team that needs to quickly come up to speed on Java."
Keson Khieu, R&D Project Manager, Hewlett Packard

FREE Java Quick Reference Card

Receive a *free* copy of our exclusive Java Quick Reference Card containing valuable programming information including an overview of Java syntax, Java keywords, an operator precedence table, exception handling syntax and more. Visit our web site at www.petronio.com/refjr to request your free copy.

PETRONIO Technology Group
Targeted Technology Training

394 Lowell Street, Suite Two ■ Lexington, MA 02173 ■ 781-778-2000 ■ FAX 781-778-4000 ■ www.petronio.com ■ info@petronio.com



Steven Hunt/Image Bank

JavaHelp for the Technical Writer

Mark Fisher

JavaHelp's facilities can be used with minimal effort or you can customize them to provide highly specialized intergrated help

JAVAHELP IS A new facility from Sun which provides help with Java applications and applets. Today, Web browsers are commonly used to provide help viewers for applications. JavaHelp provides a browser-like help viewer but *without* the browser. Because no browser is used, there are obviously no browser dependencies. There is no need to consider the widely differing capabilities of the browsers that customers might be using or worry about individual browser quirks. JavaHelp is written entirely in Java, so you can provide consistent help across all platforms. You are in full control of the help environment. Furthermore, the capabilities of JavaHelp can be extended—it *is* Java after all! For example, you can extend your help to directly *interact* with your application. The only limit is your imagination.

More specifically, JavaHelp is:

1. A new set of Java Application Programming Interfaces (APIs).
   - The APIs are standard extensions to the Java Developers Kit (JDK).
2. A `HelpBroker` class created using these new APIs.
   - Applications use the `HelpBroker` as an agent to invoke the display of JavaHelp's browser-like help viewer.

Figure 1.

Default help viewer, invoked through hsviewer utility.

- You can create a custom HelpBroker (and other custom classes) for specialized help needs.

3. A set of file formats for creating the help content files that will be displayed via the HelpBroker.

- The help content files are commonly called the HelpSet, where the HelpSet is a collection of files in various formats (HTML and XML, for example).

JavaHelp is a *delivery* system, not an *authoring* system. JavaHelp specifies the formats of the files that go into the HelpSet, but it does not help you create those files (with the exception of the search database). Help-authoring tool vendors have announced plans to provide tools to create the help files—Blue Sky, CreativeSoft, ForeFront, Quadralay, Virtual Media, and WexTech among them. It is also possible to create the files by hand.

Listing 1.

myHS.hs HelpSet file.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE helpset
 PUBLIC
 "-//Sun Microsystems Inc.//DTD JavaHelp HelpSet Version 1.0//EN"
 "http://java.sun.com/products/javahelp/helpse_1_0.dtd">
<helpset version="1.0">
    <title>   JavaHelp Sample                          </title>
    <maps>
       <homeID> about                                  </homeID>
       <mapref  location="myMap.map"                   />
    </maps>
    <view>
       <type>  javax.javahelp.TOCView          </type>
       <data>  myTOC.xml                               </data>
       <name>  TOC                                     </name>
       <label> Table Of Contents                       </label>
    </view>
    <view>
       <type>  javax.javahelp.IndexView    </type>
       <data>  myIndex.xml                    </data>
       <name>  NDX                               </name>
       <label> Index                             </label>
    </view>
       <view>
       <type>  javax.javahelp.SearchView          </type>
       <data>  JavaHelpIndex                             </data>
       <name>  SCH                                       </name>
       <label> Word search                               </label>
    </view>
</helpset>
```

PERSPECTIVES

There are three widely differing perspectives on JavaHelp, depending on whether you are:

1. A technical writer
2. A Java application programmer
3. A JavaHelp custom programmer

I deal primarily with what the technical writer needs to know. Following is a brief preview of each role.

Technical Writer

The technical writer creates the help files—collectively called the HelpSet. The HelpSet consists of:

1. HTML content files—the actual help information viewed by the user.
2. An XML-based map file which provides a symbolic name for the actual HTML files, names, and locations.
3. An XML-based table of contents.
4. An XML-based index.
5. Search database files.
6. A HelpSet file that links all of the files together. Note that the "HelpSet" is the collection of all the files, and that the "HelpSet file" is a specific file within the "HelpSet" collection. The HelpSet file is sort of a "portal" to all the information in the HelpSet.

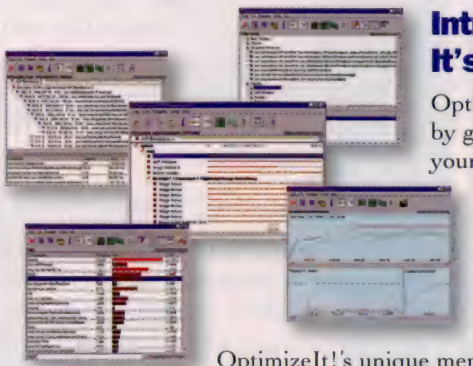
The technical writer needs to know the specifications for all these files or alternatively he/she might use an authoring tool to create some or all of them.

Java Application Programmer

The Java application programmer connects his/her application to a help viewer using a HelpBroker object. The help viewer displays the contents of the HelpSet created by the technical writer. The programmer needs to know the APIs for the HelpBroker, which are few and easy to understand. He/she needs to know relatively little about the HelpSet—just the HelpSet file name and possibly, some of the symbolic names in the map file. Optionally, the programmer could decide not to use an existing HelpBroker but could instead code the functions of a help viewer directly into his application using various JavaHelp components. He/she would need to know more JavaHelp APIs to do this.

Advanced Java profiling with OptimizeIt!

Because you have better things to do with your time than track Java performance problems.



Introducing OptimizeIt! 3.0 Professional. It's Java Profiling Like You've Never Seen.

OptimizeIt! 3.0 Professional makes your Java profiling process easier by giving you the most comprehensive and accurate information on how your Java programs use CPU and memory. What you get is clear, real-time data and easy-to-read graphs allowing you to rapidly pinpoint performance problems such as bottlenecks, excessive object allocations or memory leaks.

OptimizeIt! is the only tool to include a CPU profiler allowing you to tune CPU time measurement to the most efficient level of detail for each test session – while maintaining a low overhead. OptimizeIt!'s unique memory profiler also provides unequaled features such as a reference graph that automatically highlights the references actually causing memory leaks.

But don't take our word for it. Check us out at www.optimizeit.com to download your free trial today, and experience how OptimizeIt! will help you improve the overall performance of your Java programs. And just maybe get you out of the office on time.

www.optimizeit.com

Optimizeit 3.0 Professional supports JDK 1.1x and JDK 1.2

©1999 Intuitive Systems, Inc. All rights reserved. Optimizeit is a registered trademark of Intuitive Systems, Inc. All registred trademarks belong to their respective manufacturers.

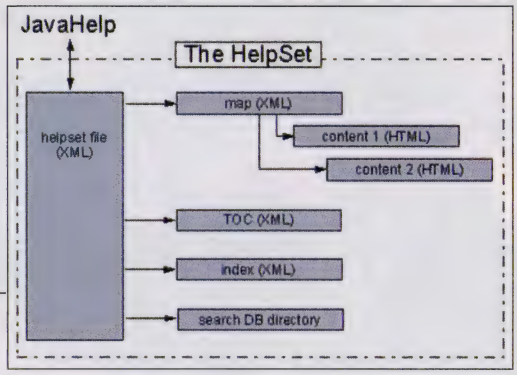


Figure 2.
The HelpSet.

JavaHelp Custom Programmer

The JavaHelp custom programmer may create a customized HelpBroker or a customized help viewer for use by the Java application programmer. He/she might also create custom navigators for use in the default or custom help viewers. The JavaHelp custom programmer therefore exploits the extensibility of JavaHelp and needs to know a wide range of JavaHelp APIs. This is the approach if the default HelpBroker or help viewer does not meet the specialized help needs of the application.

DEFAULT HELP VIEWER

The default help viewer (invoked by the default HelpBroker) is shown in Figure 1, displaying a sample HelpSet that comes with JavaHelp. The default HelpBroker can be invoked through the hsviewer utility (which also comes with JavaHelp) to visually verify a HelpSet.

The content viewer (Figure 1, right) is a JavaHelp JHelpContentViewer object, capable of displaying HTML 3.2 content files from the HelpSet. The navigators (Figure 1, left) are arranged in tabbed panes. The navigators organize in different ways the access to the HTML files. Selecting within a navigator (left) causes the corresponding HTML file to display in the content viewer (right). JavaHelp supplies a table of contents (TOC) navigator (the JHelpTOCNavigator class), an index navigator (the JHelpIndexNavigator class), and a full-text search navigator (the JHelpSearchNavigator class). Figure 1 shows a TOC navigator, and Figures 3, 4, and 5 show, respectively, a TOC navigator, an index navigator, and a search navigator.

The toolbar provides buttons to scroll backward and forward through a history of what has already been viewed, and a button to toggle the navigators on or off to expand the content viewer to fill the full width of the help viewer.

HelpSet

We now look at the HelpSet in more detail, discussing each file within the HelpSet and, where appropriate, show the resulting JavaHelp display. Recall that the HelpSet is not a file but is a collection of files. The files are created by the technical writer, possibly using an authoring tool. The relationship between the files is shown in Figure 2. The presence of some of the files is to be expected—HTML files, and, usually, some files to provide for table of contents, index, or search navigation capabilities. The remaining, less obvious files, are the map file and the HelpSet file. The map file assigns symbolic IDs for the HTML file names and locations. The rest of the files in the HelpSet (and the Java programmer) use only the symbolic IDs—there is no need to know the actual HTML file names or locations. Finally, there is the HelpSet file, which points directly (or indirectly through the map file) to all the other files in the HelpSet. The key to the HelpSet is the HelpSet file—it is the portal through which JavaHelp gets at all the information in the HelpSet, and it is the only file name that the Java programmer needs to know to use the HelpSet.

The HelpSet can be delivered as individual files or with the files packaged together into one JAR file. (A mixture of individual files and JAR files is possible, but not encouraged.) I show all the files delivered individually rather than packaged into a JAR.

HelpSet File

As noted, this file is the portal to all the other files in the HelpSet. It is coded to XML standards, with HelpSet file-specific tags. Our sample HelpSet file—myHS.hs—is shown in Listing 1.

Within the <maps> tag, the optional <homeID> tag points to the ID of the HTML file to be shown if the help viewer is made visible with no specified content (the content viewer will be blank otherwise). The ID resolves to an actual HTML file through the map file, specified in the <mapref> tag. After the <maps> tag, you will specify one or more navigational views such as a table of contents, index, or full-text search. Navigational views are optional, but for the help to make any sense, at

Listing 2.

myMap.map map file.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE map
 PUBLIC
 "-//Sun Microsystems Inc.//DTD JavaHelp Map   Version 1.0//EN"
 "http://java.sun.com/products/javahelp/map_1_0.dtd">
<map version="1.0">
    <mapID target="about"      url="about.html"          />
    <mapID target="topic1"     url="./details/topic1.html" />
    <mapID target="topic2"     url="./details/topic2.html" />
    <mapID target="comp.copy"  url="copybutton.html"     />
    <mapID target="comp.paste" url="pastebutton.html"    />
</map>
```

Now Even More AMAZING!

Are You Ready for the Next BIG THING in Java Development???

jtest! is the only tool on earth which finds your uncaught runtime exceptions for you!

But don't take our word for it. Try it for yourself!!!

Watch as **jtest!** goes through your code, automatically finding input to flush your bugs!

Marvel as it reports each exception found!

Wonder how you ever got along without it!!

jtest! works on:
Applets
Applications
API's
JavaBeans
any Java Code!

Find Hidden Bugs in Java Classes!

jtest

FREE Download

www.parasoft.com/jtest

1-888-305-0041

ParaSoft Corporation

least one of them should be present. Note that each navigator appears within a <view> tag. Within a <view> tag, the <type> tag specifies the type of navigator. The <data> tag specifies a source of data used to build the navigator. Within the <view> tag, the <name> tag should be used to give a unique identifier to each navigator view within the HelpSet. The <label> tag should be used to give some text for the Swing tooltip "fly-over" help for the navigator tabs. The HelpSet file is usually fairly short and easy to create by hand, without the help of an authoring tool.

Map File

Our sample map file—myMap.map—is shown in Listing 2.

The map file specifies a symbolic ID (the target attribute) for each HTML file (the url attribute). Note the use of "/" instead of "\" to specify the subdirectory structure of the HTML files that are not in the same location as the map file. These IDs rather than the actual HTML file names and locations are used by the rest of the files in the HelpSet and by the Java application programmer. Therefore, the actual HTML file names and locations can change without affecting anyone else. The map file could be long if there are many HTML files, but its structure is simple and it could be created by hand without the help of an authoring tool.

TOC File

The TOC file is coded to XML standards, with TOC-specific tags. Our sample TOC file—myTOC.xml—is shown in Listing 3, and the resulting TOC navigator is shown in Figure 3.

A <tocitem> tag can have another <tocitem> tag nested within it. Nesting can be to any level. A nested <tocitem> causes an indented tree structure and the appearance of a folder, as shown in Figure 3. Note the text attribute "About the application" in Listing 3 and note that the text appears in the actual TOC navigator in Figure 3. Note the use of the target attribute within the same <tocitem> tag, specifying an ID from the map file—the ID of "about" is associated with the text "About the application." Selecting the text "About the application" in the actual TOC navigator will cause the HelpBroker to display in the content viewer the HTML file associated with the "about" ID. Typically, only the lowest-level nested <tocitem> tags will have associated target attributes, but it is possible to have a target attribute at any level, as shown with the first (and highest level) <tocitem> in Listing 3. Depending on the amount of detail you want to provide, the TOC file could be long and complex, so you may want to use an authoring tool.

Listing 3.

myTOC.xml TOC file.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE toc
 PUBLIC
 "-//Sun Microsystems Inc.//DTD JavaHelp TOC | Version 1.0//EN"
 "http://java.sun.com/products/javahelp/toc_1_0.dtd">
<toc version="1.0">
    <tocitem text="About the application" target="about"      >
        <tocitem  text="Details"                              >
            <tocitem  text="Topic 1"        target="topic1"   />
            <tocitem  text="Topic 2"        target="topic2"   />
        </tocitem>
        <tocitem  text="The visual components"                >
            <tocitem  text="Copy button"    target="comp.copy"  />
            <tocitem  text="Paste button"   target="comp.paste" />
        </tocitem>
    </tocitem>
</toc>
```

Index File

The index file is coded to XML standards, with index-specific tags. Our sample index file—myIndex.xml—is shown in Listing 4 and the resulting index navigator is shown in Figure 4.

An <indexitem> tag can have another <indexitem> tag nested within it. Nesting can be to any level. A nested <indexitem> causes an indented tree structure as shown in Figure 4. Note the text attribute "copy" toward the bottom of Listing 4, and note that the text appears in the actual index navigator in Figure 4. Note the use of the target attribute within the same <indexitem> tag, specifying an ID from the map file—the ID of "comp.copy" is associated with the text "copy." Selecting the text in the actual index navigator will cause the HelpBroker to display in the content viewer the HTML file associated with the ID. Typically, only the lowest-level nested <indexitem> tags will have associated target attributes (Listing 4), but it is possible to have a target attribute at any level. Finally, note the presence of a "Find" facility in the index navigator—it finds the first occurrence of the specified word in the index navigator. Depending on the amount of detail you want to provide, the index file could be long and complex, so you may want to use an authoring tool.

Search Database

The search database consists of several files, not just one, and the HelpSet file points to the *directory* containing these files, not explicitly to any particular file. The JavaHelp search engine uses the search database to link user supplied words (beside "Find" in Figure 5) to text in the HTML files, creating a list of "hits." The user then selects a "hit" to display the associated

HTML file. Figure 5 shows the search navigator with the hits resulting from a search on the words "marsupial information." The solid dot means that both words were found in an HTML file, and the half dot means that only one of the two words was found.

JavaHelp provides a utility to create the search database files for you. You can also provide your own utility for creating customized search database files. In this case, you need to explicitly specify your own search engine to be used with the custom search database files, using the <engine> tag within the HelpSet file, within the <view> tag for your custom search facility.

The utility supplied with JavaHelp can be invoked through the jhindexer command file or by running a Java Indexer application directly, supplying the appropriate parameters. Be aware that despite "index" in these names, the jhindexer command file and Indexer application have nothing to do with the index file or index navigator. The full documentation on using these utilities can be found in Sun's JavaHelp documentation.

To use the utilities, you usually provide a config file—a list of the HTML files to be processed into the search database. (Alternatively, you can pass the utilities the top level directory of the HTML files.) Our sample config file—named search.list—is shown in Listing 5.

Use the keyword "File" followed by the name of the HTML file. Note how to specify an HTML file located in a subdirectory relative to the config file, and note the use of "/" instead of "\" to specify the subdirectory structure. The actual HTML file names and locations are specified in the config file—*not* symbolic IDs from the map file. Technically, this does not violate the purpose of the map file, because the config file is *not* part of the HelpSet—only the resulting search database files are specified in the HelpSet (through their directory). You can optionally specify a stopword list in the config file. Stopwords are words to be ignored when creating the search database. A default stopword list containing such words as "the" and "this" and so on is automatically used, and the full default list is available in Sun's documentation. If you explicitly supply your own list, as shown in Listing 5, then the automatic defaults are overridden, and you should therefore consider including the defaults within your custom list.

HTML Content Files

The HTML files are displayed within the content viewer in the right side of the HelpBroker. Currently, only HTML 3.2 is supported. You can create the HTML files on your own or use your favorite HTML authoring tool or use a JavaHelp authoring tool when they are available. Recall that the map file assigns symbolic IDs to the actual HTML file names and locations. A simple about.html file is shown in Listing 6. The content of the HTML files is not important for the purposes of this article, so no listings are provided for the remaining files topic1.html, topic2.html, copybutton.html, or paste button.html.

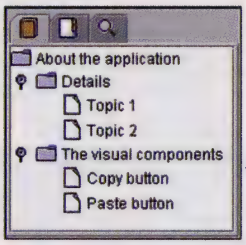


Figure 3. TOC Navigator.

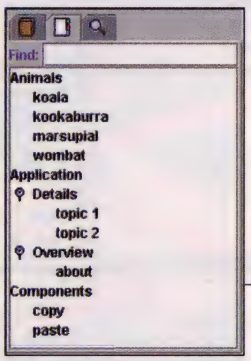


Figure 4. Index Navigator.

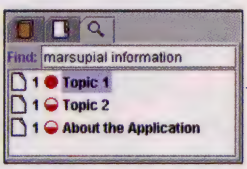


Figure 5. Search Navigator.

The HTML files are in the same location as the map file, except for topic1.html and topic2.html, which are in the detail subdirectory (see Listing 2). All the other HelpSet files are also in the same location as the map file, except for the search database files, which are in the default JavaHelpIndex subdirectory.

Listing 4.

myIndex.xml index file.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE index
 PUBLIC
 "-//Sun Microsystems Inc.//DTD JavaHelp Index  Version 1.0//EN"
 "http://java.sun.com/products/javahelp/index_1_0.dtd">
<index version="1.0">
    <indexitem  text="Animals"                    >
        <indexitem  text="koala"     target="topic2"    />
        <indexitem  text="kookaburra" target= "topic2"   />
        <indexitem  text="marsupial"  target= "topic1"   />
        <indexitem  text="wombat"     target= "topic1"   />
    </indexitem>
    <indexitem  text="Application"                >
        <indexitem  text="Details"                 >
            <indexitem  text="topic 1"  target= "topic1"   />
            <indexitem  text="topic 2"  target= "topic2"   />
        </indexitem>
        <indexitem  text="Overview"                >
            <indexitem  text="about"    target="about"    />
        </indexitem>
    </indexitem>
    <indexitem  text="Components"                 >
        <indexitem  text="copy"      target="comp.copy"  />
        <indexitem  text="paste"     target="comp.paste" />
    </indexitem>
</index>
```

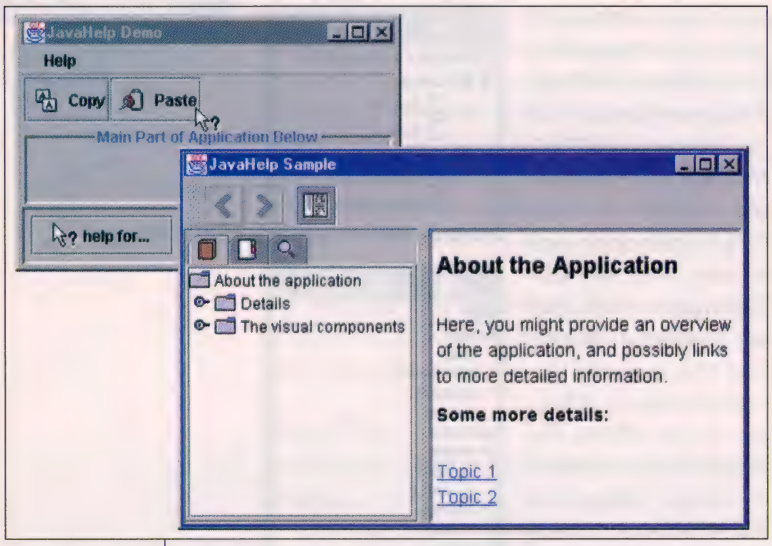


Figure 6.

Sample HelpSet invoked from sample application.

The search result (Figure 5) shows the text from the <title> tag of the associated HTML file. Therefore, the first heading tag in each HTML file should probably contain the same text as the <title> tag to provide visual consistency to the user. This is shown in Listing 6. (If a <title> tag is not found in the HTML file, the text of the first heading tag found—H1 through H6—is used in the search result.)

Figure 6 shows our sample HelpSet invoked by a sample application. Context-sensitive help is shown—the user selected the "help for..." button and the cursor has been changed to an arrow with a question mark, and a special event listener is in effect. Using the special cursor, selecting a component—such as the "paste" button shown in Figure 6—will not have the usual effect but will instead display the HTML help file of the ID associated with the selected component (this association is provided by the Java application programmer).

SUMMARY

JavaHelp uses the popular browser model for providing help, but *without* the browser—therefore, there is no need to accommodate the wide range of browser capabilities and quirks. Instead, JavaHelp is implemented through new Java classes. As a result, the help environment is consistent across all platforms, which is convenient for both the help user and the help provider. The facilities of JavaHelp can be used with minimal effort, or the Java programmer can customize JavaHelp to provide a highly specialized and integrated help for an application. JavaHelp is flexible in how it is deployed. At one extreme, all the HelpSet files and help processing are the user machine. At the other extreme, only a single HelpSet is provided, (accessible on the Web by all users), with help processing (e.g., searching through the search database) taking place on the HelpSet server. ■

Mark Fisher is an Advisory Software Engineer with IBM in Research Triangle Park, NC. He can be contacted at mfisher1@us.ibm.com.

Listing 5.

search.list config file.

```
File about.html
File ./details/topic1.html
File ./details/topic2.html
File copybutton.html
File pastebutton.html
StopWords a, and, the, this, wombat, koala
```

Listing 6.

about.html.

```
<!— about.html —>
<HTML>
<HEAD>
<TITLE>About the Application</TITLE>
</HEAD>
<BODY>
<H2>About the Application</H2>
<P>Here, you might provide an overview of the
application, and possibly links to more detailed
information.</P>
<H3>Some more details:</H3>
<A HREF="./details/topic1.html">Topic 1</A><BR>
<A HREF="./details/topic2.html">Topic 2</A><BR>
</BODY>
</HTML>
```

URLs

Register at Sun's Java Developer Connection to download JavaHelp.
developer.java.sun.com/

Find out the latest at the JavaHelp Web page.
java.sun.com/products/javahelp/

Blue Sky Software
www.blue-sky.com/

CreativeSoft
www.createsoft.com

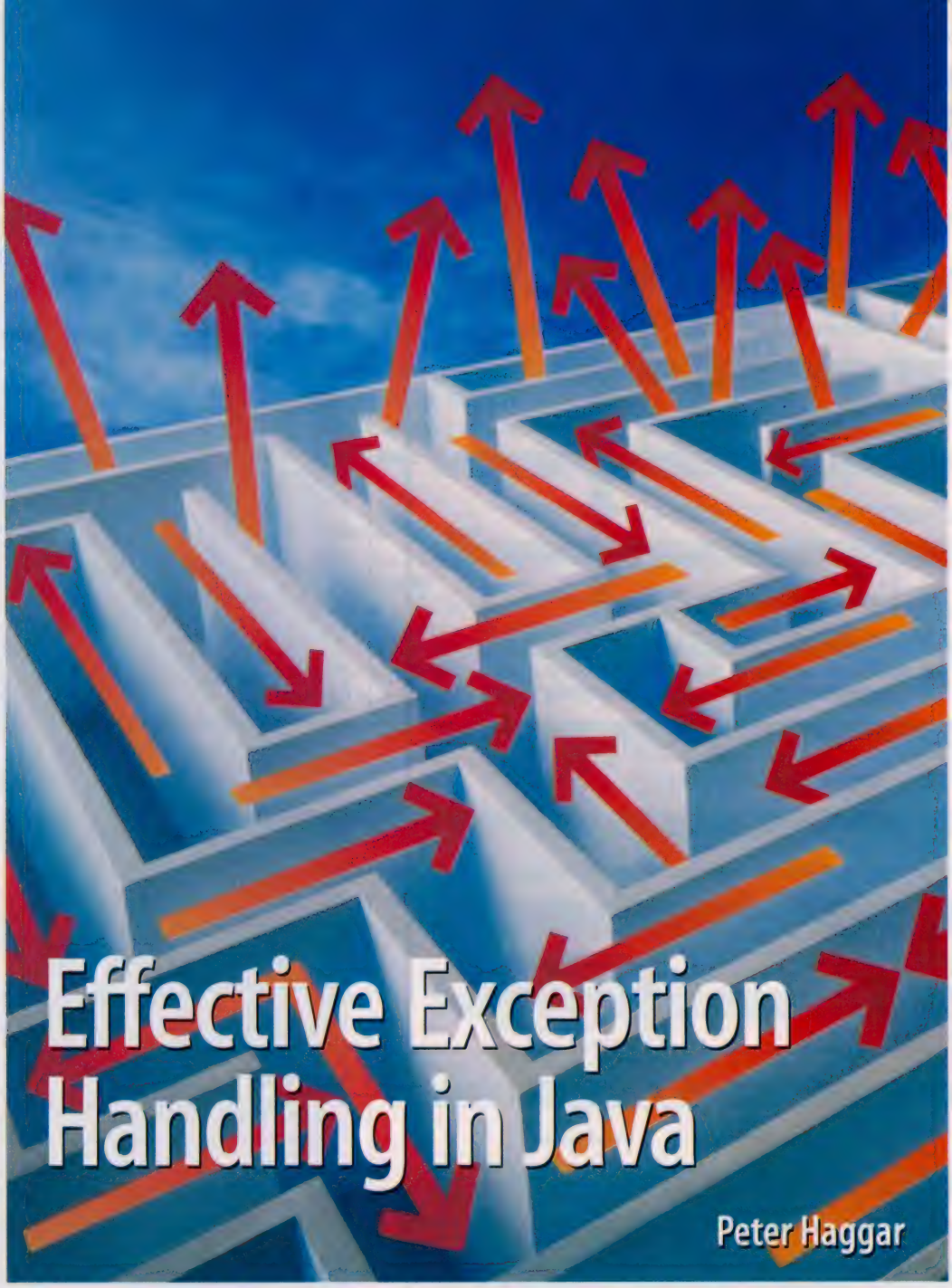
ForeFront
www.ff.com/

Quadralay
www.quadralay.com/

Virtual Media
www.virtual-media.com/

WexTech Systems, Inc.
www.wextech.com/

Quantity reprints of this article can be purchased by phone: 717.560.2001, ext.39 or by email: sales@rmsreprints.com.



Effective Exception Handling in Java

Peter Haggar

Elle Schuster/Image Bank

The most interesting and least understood areas of exception handling: *nuances* and *side effects*— when to use, when not to use, and their impact on performance

EXCEPTION HANDLING IS a very powerful and useful feature of the Java language. With this power comes a degree of complexity that must be understood to effectively utilize exception handling in Java programs. Java introduces some new concepts to the exception handling model that make exceptions both easier and more difficult to use properly.

Exception handling is not the silver bullet it is sometimes made out to be. It will not solve all your error handling problems. In fact, it will introduce new problems, unless a well-rounded understanding of this technology is grasped. Too often with exception handling, programmers are led to believe that mastering a few new keywords will solve the error handling problems encountered with traditional error handling techniques. Exception handling simply represents a different avenue for dealing with errors. While exception handling is a step forward with regard to traditional error handling techniques, it creates new issues and problems that must be dealt with. Failure to understand these issues and understand the problems surrounding them can lead to the creation of additional problems in your error handling code.

When describing exception handling, most literature stops with a cursory look at the mechanics. What is truly needed to write solid Java

ADVERTISER INDEX

Advertiser	Page	URL	e-mail
4th Pass	CV 2	www.4thpass.com	rwolf@4th pass.com
Ad Astra Engineering	29	www.JumpingBeans.com	crygaard@AdAstraEng.com
Advanced Software Technologies	1	www.advancedsw.com	sales@advancedsw.com
Alta Software	77	www.altasoft.com	hr@altasoft.com
Application Development '99	71	www.applicationdevelopment.com	enquiries@sigs.com
BEA	2	www.weblogic.beasys.com	info@weblogic.com
C++ Report	43	www.sigs.com	subscriptions@sigs.com
Cerebellum	31	www.cerebellumsoft.com	info@cerebellumsoft.com
Component Development '99	67	www.componentdevelopment.com	conferences@sigs.com
DevelopMentor	77	www.develop.com	N/A
Enterprise Software	11	www.EnterpriseSoft.com	java@EnterpriseSoft.com
Fiorano	61	www.fiorano.com	info@fiorano.com
IBM	5	www.software.ibm.com/websoftware	
Insignia	23	www.insignia.com	evm@insignia.com
InstallShield	CV 3	www.installshield.com	william@installshield.com
Intuitive	49	www.optimizeit.com	optimizeit@intuisys.com
KL Group	CV 4	www.klg.com	info@klg.com
KL Group	37	www.klg.com	info@klg.com
Numega	35	www.numega.com	carlm@numega.com
Object People	26-27	www.objectpeople.com	rausten@objectpeople.com
Parasoft	51	www.parasoft.com/jtest	joe@parasoft.com
Persistence	45	www.persistence.com	info@persistence.com
Petronio	46	www.petronio.com	info@petronio.com
Rational Software	21	www.rational.com/mvs/vismod/	info@rational.com
Rogue Wave	9	http://www.roguewave.com	shryock@roguewave.com
RST	32	www.rstcorp.com	ambruno@rstcorp.com
SD West	75	www.sdexpo.com	jdlong@mfi.com
SIGS Newsletters	73	www.javareport.com	aheck@sigs.com
Sun Hardware	40-41	www.sun.com/desktop	
Symantec	15	www.visualcafe.com	jmaloney@symantec.com
Symantec	59	www.visualcafe.com	jmaloney@symantec.com
Tidestone Technologies	19	www.tidestone.com	sales@tidestone.com
XML One	63	www.xml.conferences.com	conferences@sigs.com
Zero G	7	www.zerog.com	info@zerog.com

Java™ REPORT
THE INDEPENDENT SOURCE FOR JAVA DEVELOPMENT • www.javareport.com

BPA INTERNATIONAL

Group Sales Director
Miles E. Silverman
71 W. 23rd St., New York, NY 10010
• phone (212)242-7447 • fax (212)242-7574
• email msilverman@sigs.com

Advertising Sales Manager, Eastern Region
Brian Levine
71 W. 23rd St., New York, NY 10010
• phone (212)242-7447 • fax (212)242-7574
• email blevine@sigs.com

Advertising Sales Manager, Western Region
Jeffrey Wainhause
71 W. 23rd St., New York, NY 10010
• phone (212)242-7447 • fax (212)242-7574
• email jwainhause@sigs.com

Advertising Sales Representative for Development Solutions Buyer's Guide
Miles E. Silverman
71 W. 23rd St., New York, NY 10010
• phone (212)242-7447 • fax (212)242-7574
• email msilverman@sigs.com

ADMINISTRATION
Assistant to the President
Lynn Hazlewood

CIRCULATION
Circulation Director
Micki LaPorte

Assistant Circulation Manager
Adriel Marra

Circulation Assistant
Amit Bholasing

FINANCE
Controller
David Chatterpaul

Accounts Receivable Credit Manager
Rey Shields

Accounts Payable
Bibi Budhram

MANUFACTURING
Director of Production and Manufacturing
Elisa Varian

Assistant Production Manager
Michele Costabile

MARKETING
Director of Marketing Publications
Allyn Tracy Heck

Art Director, Marketing
Tom Jezek

Marketing Managers
Robin B. Schultz
Cheryl Cains

WORLDWIDE OFFICES
Jane Major, SIGS Ltd., Brocus House,
Parkgate Road, Newdigate, Surrey RH5 5AH, UK
• phone 011.44.1.306.631.331 • fax 011.44.1.306.631.696
• email enquiries@sigs.com

Jürgen Tillack, SIGS Conferences GmbH, Hauptstraße
293-297, D-51465 Bergisch Gladbach, Germany
• phone 49.2202.9372.0 • fax 49.2202.9372.2
• email 100634,2070@compuserve.com

code that utilizes exception handling properly is a full understanding of the mechanics of Java and then information on some of the nuances of Java exception handling. I examine the most interesting and least understood areas of Java exception handling: nuances and side effects. I also look at when to use it, when not to use it, how to use it effectively, and the impact of exception handling on the performance of your Java code.

THE NUANCES OF JAVA EXCEPTION HANDLING

All language features have their associated nuances. This is also true of exception handling in Java. Let's look at a few of them.

Keyword finally

The *finally* keyword is probably the best addition to the Java exception handling model over the C++ model. finally enables the execution of code to occur whether an exception occurred or not. Usage of finally is very good for maintaining the internal state of an object when an exception occurs as well as cleaning up non-object resources.

finally blocks are fairly straightforward, but have some behavior that can catch the unwitting developer off guard. finally blocks are always executed. More specifically, they will be entered when code within a try block:

- throws an exception
- finishes normally
- executes a return statement
- executes a break statement
- executes a continue statement

Let's look at the ramifications of this. Listing 1 contains code that has some peculiar behavior.

Executing this code reveals two obvious results and one not so obvious. This code simply calls three methods, method1(), method2(), and method3(), and prints out the values they return. The call at //3 to method1() will result in 1 being printed. The call at //4 to method2() results in 2 being printed. The catch block in method2() is never executed because an exception is not thrown. However, the call to the method3() at //5 will print 4! This is because finally blocks are always executed regardless of what happens in a try block. In this case the value 3 is about to be returned at //1. However, on the execution of the return statement, control is transferred to the finally block at //2. This causes the return 4 statement to execute, resulting in method3() returning the integer 4.

Developers think that when they execute a return statement, they immediately leave the method they are executing. This is no longer true with the usage of finally. (finally is also entered on a break or continue statement.) This particular problem has the potential to lead to long debugging sessions if you are not aware of this side effect of finally.

The throws Clause

Many view the throws clause in Java as a savior. Remember the throws clause is where you have to list all of the checked exceptions that can propagate from a method. The compiler makes sure you either catch checked exceptions in a method or declare them in the throws clause of the method. The throws clause is provided to alert all callers of your method of the exceptions that it can generate. I think the throws clause is a useful language feature as implemented and provides useful information to users of methods. However, there is a side effect to it that needs to be considered as you are designing your Java code.

What happens when you are well into your coding of a fairly large project and you add a checked exception to a low-level worker method. For the purpose of this discussion, a worker method is one that performs a common task for many other methods in your system and is called from many, many places in your code. Typical

Listing 1.

Nuances of finally.

```
class finallyTest {
    public int method1() {
        return 1;
    }

    public int method2() {
        try {
            return 2;
        }
        catch(Exception e) {return 3;}
    }

    public int method3()  {
        //Because the finally is always executed whether an exception is thrown or
        //not, this method will return 4 every time.
        try {
            return 3;  //1
        }
        finally {  //2
            return 4;
        }
    }

    public static void main(String args[])
    {
        finallyTest ft = new finallyTest();
        System.out.println("method1 returns " + ft.method1());  //3
        System.out.println("method2 returns " + ft.method2());  //4
        System.out.println("method3 returns " + ft.method3());  //5
    }
}
```

Table 1. Output from code in Listing 2.

In main, calling nestingTest()
In outer most try block
In inner try block
In inner catch block
Exception caught: Exception thrown from inner try
In inner finally's try block
In inner finally's finally block
In outer catch
Exception caught: Exception thrown from inner finally's finally block
In outer finally
In main, caught exception: Exception thrown from inner finally's finally block

systems have many such methods. You have two choices:

- Catch the checked exception in this method and handle it there.
- Throw the exception from the worker method to allow the caller to handle it.

Depending on the circumstances of your system, the first option may not be viable. This worker method may not have the means to resolve the exception itself. You may have no other choice but to choose the second option and throw this exception. This may not be as easy as it sounds.

Adding an exception to a throws clause of a method will effect *every* method that calls it. All methods that now call this worker method have to change. They also have the same two choices on what to do as listed above. If they decide to handle the exception, the exception will not propagate any further. However, what if these methods do not have the means of handling the exception? They too must rely on the second option and add the exception to their throws clause. After doing this you may attempt to compile your code again. When the compiler is done spitting out error messages, you may now realize what you are up against. Now, all methods that call all of the methods you just changed, have the same two choices. The process will continue all the way back to main(), assuming no intervening method handles the exception.

The moral of this story is not to add exception handling at the end of the development cycle. *Your error handling strategy should be designed in from the beginning.* If, as is often the case, you run into this situation, even with careful planning, you now know what you may be up against. The throws clause is a useful and beneficial language feature, but can also be painful if you're not careful.

Nested Exceptions

What happens during the processing of a catch or finally block, after an exception has been thrown, when some of the code contained within the catch or finally throws an exception? In addition, can you have a try/catch/finally block nested within a try, catch, or finally block? If so, what would be the flow through this type of code? Listing 2 contains a code example that answers these and other questions. Trace through the code and try to determine the output of the code (which is shown in Table 1).

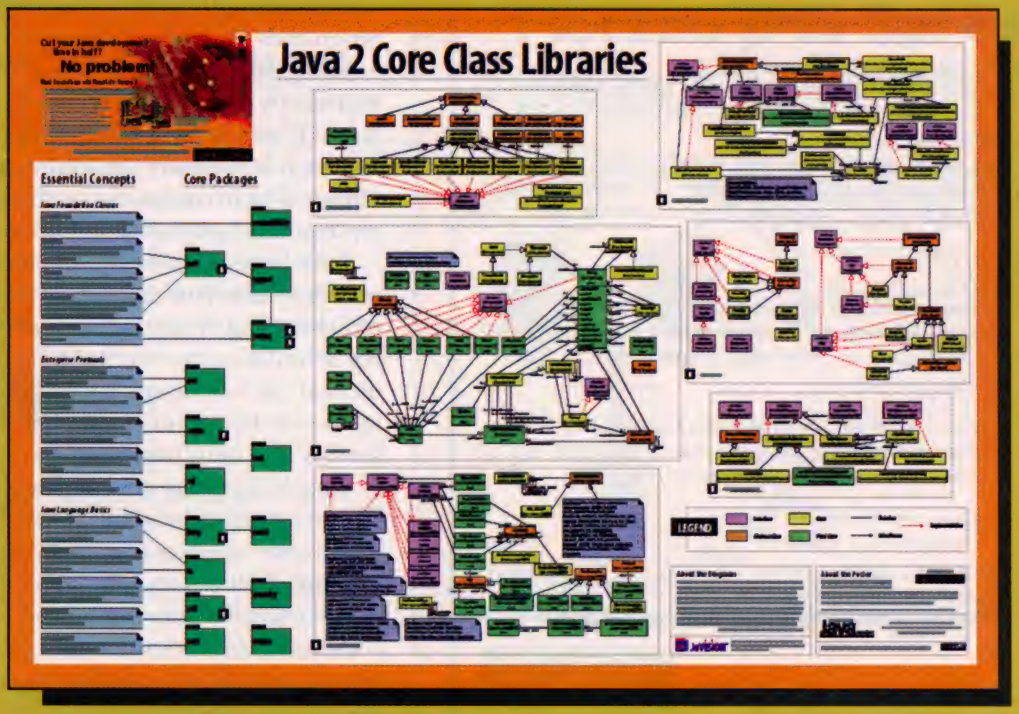
The code in Listing 2 consists of various try/catch/finally blocks nested within one another. You can place try/catch/finally, try/catch, or try/finally blocks anywhere in your Java code. In addition, as is demonstrated, these blocks can be nested within each other. A try/catch/finally block can be placed inside of another try, catch, or finally block. There are ramifications to this, the least of which is the code can become somewhat confusing. The more serious side effect is that exceptions get lost or hidden from the caller.

Listing 2 consists of the method main(), which calls a method nestingTest() and catches an exception thrown from it at //1. Method nestingTest() consists of an outermost try/catch/finally block at //2, //12, and //13. Inside of the try block at //2, we have another try/catch/finally block at //3, //5, and //7. Inside of the finally block at //7, we have an additional try/finally block at //8 and //10. Let's examine the flow through this code.

On entering main(), we print a message and call method nestingTest(). Once inside nestingTest(), we enter the outermost try block at //2, print a message, and enter the inner try block at //3 printing another message. We then immediately throw an exception at //4. When an exception is thrown from a try block, control is transferred out of the try block to its corresponding catch block. If a catch block doesn't exist, control will transfer to the corresponding finally block. (Try blocks cannot exist without a corresponding catch or finally block.) In this case, control is transferred to the catch block at //5, and we print our message indicating the exception we caught. We then throw another exception at //6. Before we can leave this code, we must always execute the finally block. We enter the finally block at //7, and immediately enter yet another try block at //8. We print a message and throw another exception at //9. We then en-

Unfold The Secrets of the Java 2 Class Hierarchy

Want the **FREE** all new, illustrated, colorful, really great poster that details the core concepts of the Java™ 2 platform, including **FREE** descriptions of the Java™ Foundation Classes, the Enterprise Protocols, and the Java™ Language Basics? It's filled with visual representations of the Core Class Libraries with selected diagrams representing the most important new Java™ 2 Classes. Did we mention that it's **FREE**? From Symantec™. Of course you want it. Come to www.javareport.com/poster so we can send one right to you. If you want, we'll also send you a **FREE** issue of **Java™ Report** just for stopping by.



SYMANTEC.

www.javareport.com/poster

ter the finally block at //10, print a message, and throw yet another exception at //11.

So far, four exceptions have been thrown, one each at //4, //6, //9, and //11 and we have completed executing the outer try block at //2. It is interesting to note what happens next. We enter the catch block at //12, print a message indicating the exception we caught, then rethrow the exception and enter the outermost finally at //13 and print a message. When the finally block at //13 finishes, control is returned to the catch block in main() at //1. Note that the exception we catch at //1 is the exception thrown from //11. What happened to the exceptions thrown from //4, //6, and //9? The exception thrown at //4 was caught at //5 and not rethrown. The exceptions thrown from //6 and //9 were not caught by any code, but what has happened to them? In short, they are lost. They are also referred to as hidden. The exception thrown at //9 hid the one thrown at //6. The exception thrown at //11 hid both of them. What are the effects of hiding exceptions? For one, the original exception is lost and the code processing the last exception thrown (//12 in Listing 2) will not know about the previous exceptions thrown. This could be very bad, because the original problems reported in the exceptions thrown at //6 and //9 are not known.

Losing or otherwise hiding exceptions is not a good thing. What can or should you do about it? The best way to deal with this is when throwing exceptions from the inside of a try, catch, or finally block, wrap the existing exception inside of any new exceptions you may throw. This way, the receiver of the new exception will have information about all errors, and critical error information will not be lost.

The code in Listing 2 shows us that try/catch/finally blocks can be arbitrarily nested. In addition, only one exception can propagate from a try/catch/finally block even though many can be thrown from within it. Remember that only the last exception thrown will be seen by the caller while others will be hidden and lost.

Listing 2.

Nesting try, catch, and finally.

```
class finallyThrowTest {
    public static void main (String args[]) {
        finallyThrowTest ftt = new finallyThrowTest();
        try {
            System.out.println("In main, calling nestingTest()");
            ftt.nestingTest();
        }
        catch (Exception e) { //1
            System.out.println("In main, caught exception: " + e.getMessage());
        }
    }
    public void nestingTest() throws Exception {
        try { //2
            System.out.println("In outer most try block");
            try { //3
                System.out.println("In inner try block");
                throw new Exception("Exception thrown from inner try"); //4
            }
            catch (Exception e1) { //5
                System.out.println("In inner catch block");
                System.out.println("Exception caught: " + e1.getMessage());
                throw new Exception("Exception thrown from inner catch"); //6
            }
            finally { //7
                try { //8
                    System.out.println("In inner finally's try block");
                    throw new Exception("Exception thrown from inner finally's
                                        try block"); //9
                }
                finally { //10
                    System.out.println("In inner finally's finally block");
                    throw new Exception("Exception thrown from inner finally's
                                        finally block"); //11
                }
            }
        }
        catch (Exception e2) { //12
            System.out.println("In outer catch");
            System.out.println("Exception caught: " + e2.getMessage());
            throw e2;
        }
        finally { //13
            System.out.println("In outer finally");
        }
    }
}
```

WHEN TO USE AND WHEN NOT TO USE EXCEPTION HANDLING

Exception handling was devised as a robust replacement for traditional error handling techniques. Some believe that exception handling should be used for any and all error conditions and thus completely avoiding traditional error return code checking. I believe that exception handling can be overused. It should not be used exclusively or for control flow as a replacement for the if/else clause. I believe a more moderate approach is necessary whereby exception handling is used in conjunction with traditional error handling techniques.

Let's look at an example of this: Listing 3 contains some Java code that uses traditional error handling. Notice at //1 we are checking to see if the data value returned from the getData() method call is non-zero. If it is zero, we assume we are at the end of the stream. This makes sense and is intuitive.

Now let's change this code by taking a hard line on exception handling. We will not rely on return code values as is done at //1, but use exceptions. Listing 4 contains the code in Listing 3 with exception handling added.

Notice at //1 we are placing a try/catch block around the call to the getData() method. getData() will no longer return zero when the stream is empty, it will throw a NoMoreDataException. Even though the code in Listing 4 is a bit uglier than the code in Listing 3, some people be-

Listing 3.

Proper use of not using exceptions.

```
int data;
MyInputStream in = new
MyInputStream("filename.ext");
data = in.getData();
while (data != 0)  //1
{
    //do something with data
    data = in.getData();
}
```

lieve that Listing 4 represents the way code should be written with exceptions and that returning error codes should never be done. I disagree. I think Listing 3 is much more intuitive even though it relies on the older methods of dealing with errors or unexpected results. I think exceptions can be overused and Listing 4 is a good example of it.

Exceptions should be used for conditions outside the expected behavior of the code. In the example above, we expect to get to the end of the stream, therefore a simple zero return from the getData() method is appropriate and intuitive. I don't think throwing an exception in this case is wise as this is not an exceptional condition but an expected one. We would not, however, expect that the stream is corrupted. That type of condition would warrant an exception being generated. The point is not to use exceptions for all conditions, but to use them where it makes sense, where exceptional conditions exist.

EXCEPTIONS AND PERFORMANCE

There are three areas of exceptions and performance to address. The first is the effect of throwing an exception. The second area concerns the effects of having try/catch blocks in your code even when no exceptions are thrown. The third area addresses the question: Can I write faster Java code by throwing an exception? The answer to this may surprise you.

Listing 4.

Improper use of exceptions.

```
//Improper use of exceptions. Now getData() throws
//an exception when there is no more data
int data;
MyInputStream in = new
MyInputStream("filename.ext");
while(true)
{
    try {  //1
        data = in.getData();
    }
    catch (NoMoreDataException e1) {break;}
    //do something with data
}
```

Ramifications of Throwing Exceptions

As some may believe, the act of throwing an exception is not free. After all, what are Java exceptions? Java exceptions are objects and these objects need to be created. Creating objects is somewhat costly, therefore, the act of throwing an exception has some cost to it. To throw an exception in Java you write something like:

```
throw new MyException(...);
```

This creates a new object and then transfers control to one of multiple places. Because there is a cost to throwing exceptions, I recommend exceptions be used only for error conditions and not for control flow. See "Faster Code with Exceptions?" below for an exception (no pun intended) to this rule. The reason for this is due to the object creation aspect of exception handling, which thereby makes throwing exceptions inherently slow. When exceptions are used for error/failure conditions, I am not too concerned with speed. I want my code to run fast when it's *working* and normally don't care how long it takes to *fail*.

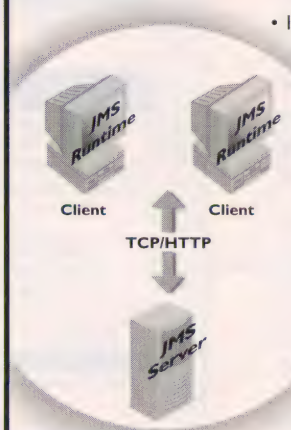
Another item to note is that the act of throwing an exception is analogous to a jump or goto statement. When an exception is thrown, pro-

Goodbye proprietary messaging!

Introducing the . . .

Fiorano JMS Server

Dramatically reduces development time for complex network applications.



- 100% pure Java implementation of JMS (Java Message Service) API's. Includes Pub/Sub, PTP, Transactions, Durable Subscribers and more.
- Comprehensive Admin. API
- Lightweight, efficient and robust environment; thin runtime to access JMS services.
- Integrates with popular Application Servers.
- Efficient Server-to-Server message routing allows massive scalability.

Download a 30-Day Free Evaluation at www.fiorano.com

Fiorano

Fiorano Software, Inc.
236 N. Santa Cruz Ave., Suite #213 • Los Gatos, CA 95030
Phone: (408) 354-3210 • Fax: (408) 354-0846
email: info@fiorano.com • www.fiorano.com

Table 2. Average times for the code in Listing 5.

Array Size	Average time for method1()	Average time for method2()
10,000	5 ms	6 ms
100,000	51 ms	59 ms
1,000,000	507 ms	588 ms

gram control is immediately transferred to one of three places:

- catch block
- finally block
- calling method

All of these options have one thing in common. The code that was executing at the time an exception is thrown stops executing and control is transferred elsewhere. This has ramifications on the state and validity of your objects.

Whenever discussing performance and Java you have to account for the Java virtual machine (JVM) you are using and the operating system you are running on. I have seen large differences between JVMs I have used. Therefore, for what I am about to discuss, I highly recommend you perform some level of profiling on your systems first before assuming you will get the same results as indicated here.

Listing 5.

Effects of try/catch blocks on performance.

```
class excperf1Test {
    public excperf1Test(int size, int iter) {
        long time1 = 0;
        long time2 = 0;
        for (int i=0; i<iter; i++) {
            time1 += method1(size);
            time2 += method2(size);
        }
        System.out.println("For " + size + " interations, normal processing took an " +
                                    "average of " + time1/iter + " milliseconds");
        System.out.println("For " + size + " iterations, try/catch block processing
          took " +
                                    "an average of " + time2/iter + " milliseconds");
    }

    public long method1(int size) {  //1
        System.gc();
        int[] ia = new int[size];
        long start = System.currentTimeMillis();

        //Assign each value of the array using a for loop that contains no try/catch
        //blocks.
        for (int i=0; i<size; i++)
            ia[i] = i;
        long end = System.currentTimeMillis();
        return (end-start);
    }

    public long method2(int size) {  //2
        System.gc();
        int[] ia = new int[size];
        long start = System.currentTimeMillis();

        //Use a for loop that has a try/catch block inside it.
        for (int i=0; i<size; i++) {
            try {  //3
                    ia[i] = i;
            }
            catch (Exception e2){}
        }
        long end = System.currentTimeMillis();
        return (end-start);
    }

    public static void main(String args[]) {
        excperf1Test t = new excperf1Test(Integer.parseInt(args[0]),
          Integer.parseInt(args[1]));
    }
}
```

Effects of try/catch Blocks on Performance

Placing try/catch blocks in your Java code can actually slow it down, even when exceptions are not thrown.

Listing 5 contains two methods, method1() and method2(), at //1 and //2, that contain almost identical code. Method1() does not contain any try/catch blocks, while method2() contains the code of method1() with the addition of a try/catch block in the body of the *for* loop at //3. method2() doesn't actually ever throw any exceptions, it just has a try/catch block surrounding code inside of its *for* loop. Running this code with JDK 1.1.4 on an unloaded Windows NT 4.0 machine produces results shown in Table 2. The average time indicated is based on 50 runs of the program with the specified array size.

We notice that method2() is about 16% slower than method1() for large arrays. Because no exceptions are being thrown, this is simply a function of method2() having a try/catch block in its version of the for loop. If the try/catch block is placed outside of the *for* loop, there is no difference in the execution time of the two methods.

In general, it is a better programming practice to place try/catch blocks outside of loops. Failure to do this can have a negative impact on the runtime performance of your code. Having said this, it is true that you will not see slower results on some JVMs. Running this code on JDK 1.2 does not show any difference in the timings. However, you may not know which JVMs your customers will be running, so it is still a good idea not to place try/catch blocks inside of loops.

Introducing

XML ONE

The International Users Conference

3 Tracks, 33 Sessions, 333 Answers

Stay Ahead Of Your Competition

Today's expert programmers agree- XML is the choice technology for exchanging structured information over the Internet, Intranets and Extranets. **XML One** was created out of the growing need for a developer/programmer/designer oriented industry-wide forum on this emerging technology. So, we've gathered together the top experts in the field and put together 4 action packed days of non-stop learning on the topics you need to know. The 33 fast-paced sessions – from pre conference tutorials to night school to the 3-day conference – and a 2-day jam-packed expo, round out your total XML learning experience.

CONFERENCE AT A GLANCE

seating is limited

Monday, May 24, 1999

9:00 - 4:00	M 1 XML for Java Programmers — Christian Horn	M 2 Python and XML — Paul Prescod	M 3 XML Syntax Nitty Gritty — Tommie Usdin; M 4 An In-Depth Intro to XFDL — Chris Dudenas
6:00 - 8:00	N 1 XML Middleware — Max Dolgicer	N 2 Using XML in Data Driven applications: a project insight — Mathias Lippert	N 3 Beyond DTDs: Current Proposals and the Future of Schema Definition — Bob Ducharme

Tuesday, May 25,1999

	XML ON THE WEB	XML ON THE ENTERPRISE	IMPLEMENTING XML BASED SOFTWARE
8:30 - 8:50	Opening Address: Technical Chair, Kevin Dick		
9:00 - 11:00	T 1 XML FAQ: Introduction to XML on the Web — Pamela Campbell	T 2 XML Enabling the Seamless Enterprise — Kevin Dick	T 3 The Why's of the XML Family of Specifications — Dave Orchard
11:15 - 12:00	Keynote, Simon Phipps, IBM		
12:00 - 7:00	Exhibit Open		
12:00 - 3:00	Break		
3:00 - 5:00	T 4 XML Web Sites the Microsoft Way — Scot Wingo	T 5 The Ladder of Integration: The How To's of Linking Business with XML — Charles Allen	T 6 XML and Java: Toward Data Driven Application Frameworks — Christian Horn
6:15 - 7:00	Reception & Expo		

Wednesday, May 26,1999

	XML ON THE WEB	XML ON THE ENTERPRISE	IMPLEMENTING XML BASED SOFTWARE
9:00 - 11:00	W 1 Components, XML and DHTML- the Perfect user Interface — Christian Gross	W 2 Harnessing the Power of Server-Side XML — Coco Jaenicke	W 3 Java in XML — Daniel Zen
12:00 - 2:30	Break		
12:00 - 5:00	Exhibit Open		
2:30 - 4:00	W 4 Designing Mark-Up for Precision Retrieval — Tommie Usdin	W 5 Facilitating Knowledge Engineering with XML — Kurt D. Fenstermacher	W 6 Secrets to Using Object Databases with XML — Paul Prescod
4:15 - 5:45	W 7 Extending Business Process Boundaries with XML & the Web — Tarun Sharma	W 8 XFDL - Representing Business and Government Forms in XML — Chris Dudenas	W 9 Understanding the Document Object Model (DOM)' — Reaz Hoque
6:00 - 8:00	N 4 Java, XML and DOM Object Models — David Orchard	N 5 XML and EAI — Chris Kurt	N 6 Distributed Object Architectures with XML — Henry Balen

Thursday, May 27,1999

	XML ON THE WEB	XML ON THE ENTERPRISE	IMPLEMENTING XML BASED SOFTWARE
9:00 - 11:00	Th 1 Problem Solving: Agents and XML — Christian Gross	Th 2 SWAP: Simple Workflow Access Protocol — Surendra Reddy	Th 3 XML Development Process in Java — Jason Purdy
12:00 - 1:00	Break		
1:00 - 3:30	Th 4 Use of XML at Canadian Government — Chris Bennett	Th 5 Streamlining the Product Documentation Process with XML — Tom Jell	Th 6 Making Intelligent XML Documents: A Case Study — Dave Manning
4:00 - 5:30	Th 7 Hyperwizard: An XML Scripting Language on Top of CGI — Neil Kipp	Th 8 Challenges of Managing XML Documents — Fernando Velez	Th 9 Developing a Dynamic XML Server — Bob Bickel

For information contact:

conference registration: Jonathan Goldberg, at (212)242-7515; or by email at jgoldberg@sigs.com.
exhibiting: Dennis Leavey, at (212)242-7447 ext. 136; or by email at dleavey@sigs.com

Conference
May 24-27 1999

Exhibition
May 26-27, 1999

Austin Marriott at the Capital
Austin, TX

The "how-to" XML event

What is XML & Why is it Important?

XML stands for extensible markup language and is a data format for structured document interchange on the Web. Like HTML, it is derived from SGML. But unlike HTML, which was created to allow cross-platform formatting of information for display, XML is not simply for organizing data. One of XML's primary uses is information exchange between different and potentially incompatible systems.

In short, XML is being used to provide a format for specification for the interactions between business applications. This standard provides an open framework for integration, supported by emerging industry standards, 3rd party solutions, and quickly expanding resource pools.

Technical Chair

Kevin Dick is the author of the forthcoming book, XML: A MANAGER'S GUIDE, and editor of ENTERPRISE JAVA. His consulting company, Kevin Dick Associates, has helped dozens of organizations evaluate opportunities for leveraging advanced technologies like XML, Java, and the Internet.

Presented by: SIGS CONFERENCES

Sponsored by: IBM

Table 3. Average times for the code in Listing 6.

Array Size	Average time for method1()	Average time for method2()
10,000	5 ms	4 ms
100,000	50 ms	42 ms
1,000,000	497 ms	410 ms

Faster Code With Exceptions?

After what we have seen so far, you may not think it's possible to write Java code that utilizes exceptions that is faster than Java code that does not. There are actually cases where this is possible. Let's consider the code in Listing 6.

Listing 6.

Java code utilizing exceptions to gain faster performance.

```
class excperf2Test {
    public excperf2Test(int size, int iter) {
        long time1 = 0;
        long time2 = 0;
        for (int i=0; i<iter; i++) {
            time1 += method1(size);
            time2 += method2(size);
        }
        System.out.println("For " + size + " interations, normal processing took an
          average of " +
                                    time1/iter + " milliseconds");
        System.out.println("For " + size + " iterations, exception processing took an
          average of " +
                                    time2/iter + " milliseconds");
    }

    public long method1(int size) {  //1
        System.gc();
        int[] ia = new int[size];
        long start = System.currentTimeMillis();

        //Assign each value of the array using a standard for loop construct.
        for (int i=0; i<size; i++)  //2
            ia[i] = i;
        long end = System.currentTimeMillis();
        return (end-start);
    }

    public long method2(int size) {  //3
        //Now perform the same assignment as method1, but do not use a standard
        //for loop. Use the ArrayIndexOutOfBoundsException as the loop
        //termination.
        System.gc();
        int[] ia = new int[size];
        long start = System.currentTimeMillis();
        try {
            for (int i=0; ; i++)  //4
                ia[i] = i;
        }
        catch(ArrayIndexOutOfBoundsException e)  //5
        {
            long end = System.currentTimeMillis();
            return (end-start);
        }
    }

    public static void main(String args[]) {
        excperf2Test t = new excperf2Test(Integer.parseInt(args[0]),
          Integer.parseInt(args[1]));
    }
}
```

Listing 6 contains two methods, method1() and method2(), at //1 and //3, that contain almost identical code. Method1() at //1 creates an array of integers, then initializes each value in the array from 0 to the size of the array minus 1. Notice at //2, the loop to perform the assignments is a typical *for* loop in Java. It starts at 0, increments by 1, and terminates when the counter becomes equal to the size of the array. The time it takes to complete this operation, in milliseconds, is then calculated and returned.

Let's compare this to method2() at //3. Method2() performs the exact same operation on an array, but the *for* loop is different. Notice at //4 the *for* loop is coded to run indefinitely. It still starts at the value 0 and increments by 1, but it does not check when the counter becomes equal to the size of the array. Instead, this code waits for the ArrayIndexOutOfBoundsException to terminate the loop at //5. The time it takes to complete this operation, in milliseconds, is then calculated and returned.

Running this code with JDK 1.1.4 on an unloaded Windows NT 4.0 machine produces the results shown in Table 3. The average time indicated is based on 50 runs of the program with the specified array size.

Table 3 indicates that the time to execute the loop in Listing 6 at //4 is about 17% faster than the loop at //2 for large arrays. What is going on here? We just discussed how throwing exceptions is expensive and having try/catch blocks in your code can slow it down. Given this, how can the code in method2() in Listing 6 take less time to execute than the code in method1()?

The main reason is because the *for* loop in method1() at //2 performs two checks each time through to see if it has reached the end. The first check is done with the i<size; statement of the *for* loop. The second check is performed by the Java runtime to see if the array access is out of bounds or not. The *for* loop in method2() at //4 eliminates the first check and relies on the Java runtime to perform the array bounds check. The Java runtime will throw the ArrayIndexOutOfBoundsException if you attempt to access beyond the bounds of an array. Therefore, the loop in method1() performs two checks: one in the loop and the other in the Java runtime. The code in method2() only

performs one check in the Java runtime because its *for* loop does not check an end condition. When the loop has reached beyond the end of the array, the Java runtime throws the ArrayIndexOutOfBoundsException. This exception is caught at //5 and the loop is terminated. Also note that the try/catch block is placed *outside* of the *for* loop. This is crucial for the performance gain.

Note, regarding the technique used in method2() of Listing 6—I do not recommend using this technique casually. It can be confusing and is not a standard way to program. Using it only when you have a very performance-critical area of code. In addition, not all JVMs exhibit the same timing behavior. I ran the same code on JDK 1.2 and saw no difference in the timings. This is due to the different optimizations used in JVMs. Therefore, I highly recommend before using this technique that you profile carefully to determine that this actually results in faster code on the JVM you are going to run on. Also, if you use this trick, comment it well for the person who may modify the code at a later time.

EFFECTIVE EXCEPTION HANDLING

All of the above information is useful, but I don't want to lose sight of what you are trying to accomplish when an exception is thrown. Throwing an exception is easy. The hard part is minimizing the damage you can cause by throwing one.

Object State Management

What good is it to throw an exception if you leave an object in an invalid or undefined state? After all, the code catching the exception you throw may handle it and recover and call your code again. If your object was left in a bad state, there is a chance that the code will fail. This situation can be more difficult to debug than if you just terminated the program on the first exception instead of trying to recover from it. When throwing exceptions, it is important to consider what state your object is currently in. If it's in an invalid state, consider what has to be done to place it in a valid state before throwing the exception. Consider the code in Listing 7.

Listing 7 contains an add() method, which adds an object to some list. The first thing it does is increment a counter at //1 for the number of objects in the list. It then conditionally reallocates the list and adds the object to the list at //2. The code in Listing 7 is flawed. Notice that the code between //1 and the addToList() call at //2 can throw exceptions. If an exception is thrown after //1, the object is now in an invalid state because the counter, numElements, is incorrect. If the caller of this method recovered from the thrown exception and called this method again, there is a good chance that other problems will arise because the object state is invalid.

The fix for this problem is a simple one and is shown in Listing 8. We simply move the increment of numElements to the end of the method at //1. This ensures the counter, numElements, is accurate because we increment it after we have successfully added the object to the list. This is a simple example, but it exposes a potentially serious problem. You need not only worry about the object you are currently running in, but other objects you may have modified in your method. You may have completely or partially created a file, opened a socket, or made remote method calls to another machine when you threw an exception and exited the method. You need to be aware of the object states that exist on your local machine, or a remote machine, if you expect your code to work properly when reentered at a later time.

If you expect the callers of your code to recover from the exceptions you generate, you should review all of the places where you generate exceptions to see if you are leaving your objects in a valid state. If you are not, you need to ensure that if your code is entered again, it will work properly. The finally keyword can and should be used to help preserve object state.

continued on page 77

Listing 7.

Code that leaves an object in an invalid state when throwing an exception.

```
int numElements;
public void add (Object o) throws SomeException
{
    //...
    numElements++; //1
    if (myList.maxElements() < numElements)
    {
        //reallocate myList
        //copy elements as necessary
        //could throw exceptions
    }
    myList.addToList(o); //Could throw exception //2
}
```

Listing 8.

Correct implementation of Listing 7.

```
int numElements;
public void add (Object o) throws SomeException
{
    //...
    if (myList.maxElements() == numElements)
    {
        //reallocate myList
        //copy elements as necessary
        //could throw exceptions
    }
    myList.addToList(o); //Could throw exception
    numElements++; //1
}
```

Your own private Idaho()

Steven W. Disbrow
diz@chattanooga.net

Steven Disbrow is the owner of EGO Systems, a computer consulting firm in Chattanooga, TN. In the past, he has worked as a freelance programmer, Webmaster, editor, writer, and technical instructor. His most recent achievement was the creation of the Advance Shipping Notice Web site for the Tennessee Valley Authority.

ONE OF JAVASCRIPT'S greatest strengths, and one of its most overlooked features, is its object-oriented nature. While almost all JavaScript programmers make use of JavaScript's built-in objects (windows, forms, etc.), very few take the time to learn how to create JavaScript objects of their own. In fact, many programmers regard JavaScript as something of a "toy" language, unfit for serious programming.

Perhaps one reason for this is that JavaScript isn't truly *object-oriented*. At best, JavaScript can be called *object-based*. It has no inheritance or true user-defined classes, but it does implement rudimentary objects with instance variables and methods.

While this may sound like a rather uninspired object-oriented implementation, it's actually quite powerful and very well suited for the Web page environment. This month I show you how to create your own JavaScript objects for use on your Web pages. In future articles, I'll show you how to tie those objects to HTML controls to give your Web pages more functionality and flexibility.

PARTS OF A PROGRAMMER-CREATED JAVASCRIPT OBJECT

The JavaScript objects you create will have three parts:

1. **Constructor Function**—This function will be called to create a new object of this type. This function must set up any instance variables and methods that the object will use.
2. **Instance Variables**—As with other OOP languages, these are the variables that will be created for each instance of this type of object.
3. **Methods**—As with other OOP languages, these are the functions that will be used by this type of object to work with its data.

Listing 1 shows a *very* simple programmer-created JavaScript object. In this example, the function "anObject()" is a constructor function that will be used with the "new" keyword to create objects of type "anObject." For example, the statements:

```
var myObject = new anObject();
var myOtherObject = new anObject();
```

will create two new variables of type "anObject." Both of these objects have an instance variable named "greeting" and they each have a method called "sayHello."

THE "THIS" KEYWORD

Inside our constructor function, you'll notice the "this" keyword. The "this" keyword simply refers to the object that is being created by the "new" keyword. The names that appear after the "this" keyword tell JavaScript what the names of our instance variables and methods should be. (Instance variable and method names are also sometimes referred to as *slots*.)

So, in our example, we are saying, "Inside *this* object that I am creating *right now*, I want two slots for instance variables and/or methods. The first slot will hold an instance variable called 'greeting.' And the second slot will be a method called 'sayHello'."

INSTANCE VARIABLES AND METHODS

What's the difference between defining an instance variable and a method? First, let's look at the line that defines our instance variable slot:

```
this.greeting = "Hi there!";
```

Introducing

Component DEVELOPMENT 99 EUROPE

Presented by SIGS CONFERENCES

Sponsored by OBJEKTspektrum Java SPEKTRUM

May 31–June 2, 1999

Arabella Sheraton Congress Hotel, Frankfurt, Germany

Get The Straight Facts

In response to the burgeoning need within the industry for practical information and education on component development, SIGS Conferences in Europe announces **Component Development Europe '99.** This three-day event has been designed to provide in-depth education on the effective implementation of component-based development. Delegates will have extensive opportunities for networking with their peers at breakfasts, lunches and receptions scheduled throughout the event.

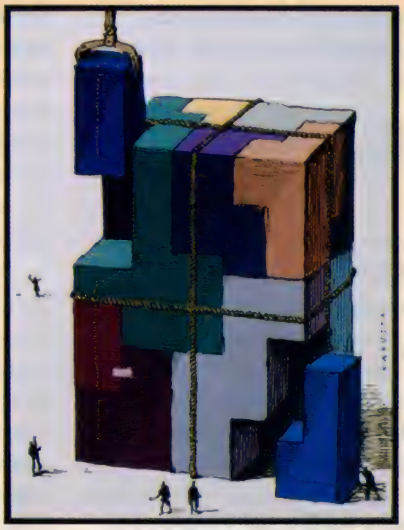
3 Days of Nonstop Learning

Be a part of this hard-core group of senior developers, project managers, systems analysts and software engineers who are serious about putting components to work in their corporations. Learn the latest component applications by attending the technical sessions scheduled over three information-packed days. Each session is geared towards intermediate to advanced-level software professionals. Get the inside scoop on the latest tips, techniques and strategies while enjoying the comfortable facilities at the Arabella Sheraton Congress Hotel in Frankfurt, Germany.

Undertake New Challenges

Topics cover a variety of component models, from COM/DCOM, to CORBA, to Enterprise JavaBeans. You'll learn how components are used throughout the development life cycle—from requirements definition, to application delivery, to runtime architecture, and from maintenance to reuse. Come and find out why components are now an evolution beyond objects–incorporating the best of objects, and adding new emerging concepts such as separation of interface and implementation, and enabling easier development through the provision of rich-runtime services.

Succeeding with Components



What You'll Learn

The program consists of 4 concurrent tracks incorporating the most advanced information from experts in the US and Germany. Meet the vendors who are pioneering the paradigm and learn from them. Classes will presented in English and German and will be technical (and practical) in nature. Over 20 sessions in all. Topics covered include:

- Conducting business and requirements analysis
- Transition management
- Selecting and integrating platforms
- Planning and managing team roles for CBD
- Building n-tier system architectures
- Component modeling and the UML
- Components on the web
- Creating and managing component repositories
- Techniques for using CORBA, COM and JavaBeans
- COM and DCOM programming using ATL
- Testing component systems
- How to select the best middleware platform
- Harnessing the power of CORBA and IIOP
- Getting the most out of EJB
- Developing a component Framework with CORBA
- Understanding MTS
- Using CORBA portable object adapter framework
- Using Java and CORBA to develop component based enterprise applications
- Building business component architecture with EJB and CORBA
- Understanding COM Architecture

Technical Chairs

Michael Stal is co-author of PATTERN-ORIENTED SOFTWARE ARCHITECTURE - A SYSTEM OF PATTERNS, and Editor-in-Chief of JAVASPEKTRUM.

Ulrich Eisenecker is an Information technology Professor at the Advanced Technical College in Heidelberg, and editor of OBJEKTspektrum's regular column "KOMPONENTEN Forum."

To register contact SIGS Conferences GmbH at:

T: +49(0)2202-9372-0 • **F:** +49(0) 2202-9372-2 • **E:** infogmbh@sigs.com

Online: www.componentdevelopment.com

Notice that on this line, we are assigning a simple scalar value (a string) to the slot. By assigning a scalar or an object to a slot, you tell JavaScript that the slot is, in fact, an instance variable.

Methods are a bit more complicated. The last line of our constructor function creates a method for our object:

```
this.sayHello = doSayHello;
```

JavaScript knows that you want "sayHello" to be a method because "doSayHello" is *not* a simple scalar or an object, it's the name of a *function* that you will define elsewhere in your JavaScript code. This function (which does *not* have to have the same name as the method it's associated with) will *implement* the method that this line defines. In other words, when the "sayHello" method gets called, the "doSayHello" function is what actually gets executed.

That function is defined on the next few lines:

```
function doSayHello() {
    alert( this.greeting)
    }
```

Notice again the use of the "this" keyword. In this case, "this" refers to the object that actually invoked the function. So, the statement:

```
myObject.sayHello();
```

will call the "doSayHello" function. The "doSayHello" function will then display the instance variable "greeting" from "myObject."

METHODS AND PARAMETERS

Parameters are passed to methods in the standard way, but methods that accept parameters are *defined* in a slightly odd fashion. When you are defining a method that takes parameters, you do *not* list those parameters in the constructor function. Instead, you list them in the implementation function.

Listing 1.

A very simple JavaScript object.

```
// This entire function is a "constructor" function.
//  It will be called in conjunction with the "new"
//  keyword to create an instance of the object type
//  "anObject".
function anObject() {
this.greeting = "Hi there!";   // Instance variable
this.sayHello = doSayHello;   // Method definition
}

// This function implements the method sayHello
function doSayHello() {
    alert( this.greeting)
    }
```

Listing 2 shows our simple object after changing it so that the "sayHello" method takes a parameter. Using this method definition, we must now pass a name string to the "sayHello" method when we invoke it. However, notice that the parameter information does not appear anywhere in our constructor function! Instead, the parameter information only appears in the implementation function for the method, the "doSayHello" function.

So, if we executed the following statements:

```
myObject = new anObject();
myObject.sayHello( "Steven");
```

An alert window would appear containing the string:

"Hi there! Nice to meet you Steven!"

DIFFERENCES FROM OTHER OBJECT-ORIENTED LANGUAGES

While the means of creating an object and its methods are slightly different from other OOP languages, the remainder of JavaScript's OOP implementation should be familiar. As we've already seen, methods and instance variables are referenced using simple dot-notation (i.e., objName.methodName(); or objName.variableName;).

Perhaps the biggest difference between JavaScript and other OOP languages is that JavaScript objects have no inheritance mechanism. This makes it rather difficult to reuse objects as the basis for new objects. About the best you can do is to include your old object in the definition of your new object as an instance variable. For example, if you wanted to create a new object "based" on our sample object, you might code a constructor function that looked like this:

```
function aNewObject() {
    this.baseObj = new anObject();
    this.var1 = 0;
    }
```

This will let you create a new object that has an instance of our old object inside of it. That object's instance variables and methods will then be available through standard dot notation. For example:

```
myNewObj = new aNewObject();
myNewObj.baseObj.sayHello("Steven");
```

HOW TO CREATE A STACK OBJECT IN JAVASCRIPT

One of the more common examples of JavaScript programming is how to create an Array object. So, I'm going to forgo that and give you a slightly more complex example of how to use what we've just looked at: how to create a

Listing 2.

A method that takes a parameter.

```
// This entire function is a "constructor" function.
//  It will be called in conjunction with the "new"
//  keyword to create an instance of the object type
//  "anObject".
function anObject() {
this.greeting = "Hi there!";   // Instance variable
this.sayHello = doSayHello;   // Method definition
}

// This function implements the method sayHello
function doSayHello( name) {
alert( this.greeting + " Nice to meet you " + name +
"!")
}
```

stack in JavaScript.

A simple *stack* is one of the most useful objects to have in your programming toolbox. Like many languages, JavaScript doesn't have a built-in stack data type. Even worse, JavaScript doesn't support pointers of any kind. So, at first glance, adding a stack data type to JavaScript might appear difficult. However, if you can envision a stack in terms of *objects* sitting on top of one another, it's easy to add a stack object to JavaScript. But, instead of a single type of object, we'll need two: a stack element object and a stack object.

THE STACK ELEMENT OBJECT

Our stack will be made up of individual elements that we will "stack" on top of each other. Each of these individual elements is actually an object, the constructors for which are shown in Listing 3 (available at *Java Report Online*).

The instance variables and methods in the "oStackElement" object are straightforward:

- **data**—This is the actual data that this stack element holds. This can be a simple scalar value or another object. (It could even be another stack object!)
- **next**—This holds the next "oStackElement" object in our stack.
- **setData**—This method sets the "data" instance variable for this "oStackElement" object.
- **getData**—This method returns the "data" instance variable for this "oStackElement" object.
- **setNext**—This method sets the "next" instance variable for this "oStackElement" object.
- **getNext**—This method returns the "next" instance variable for this "oStackElement" object.

(The methods are very simple, so we won't look at them here. However, the source code is shown in Listing 3.)

THE STACK OBJECT

After we've defined what our stack elements look like we need to define our stack object. The code for this object, the "oStack" object, is also shown in Listing 3.

This is a simple object, with just two instance variables and three methods.

- **top**—This instance variable will hold the object that sits at the top of our stack.
- **length**—This instance variable tells us how many objects are in our stack.
- **isEmpty**—This method tells us if the stack is empty.
- **push**—This method pushes a new object onto the top of the stack.
- **pop**—This method pulls the top object off of the stack and returns it.

The "isEmpty" method is very straightforward. It simply checks the "length" variable to see if it is zero. If it is, the stack is empty; otherwise, the stack is not empty.

The "push" and "pop" methods (shown in Listing 3—see *Java Report Online*) are a bit more interesting.

When we "push" an item onto our stack, the first thing we do is create a new "oStackElement" object to hold it. Then we place our data in this new object using the "setData" method. We then take the current top of the stack and assign it to the "next" field of the new item using the "setNext" method. Finally, we make our new item the top of the stack, and increment the "length" variable to show that our stack has increased in size by one element.

Popping an element off the stack is even simpler. All we need to do is capture the current top of the stack in a temporary variable and then reset the top to the next stack element (using the "getNext" method). After this is done, we decrement the "length" variable and return the former top element to the caller.

Note that when we return this top element, it's an "oStackElement" object! It's up to the caller to retrieve the data out of this object.

So, how do we use this object in a Web page? Listing 3 (available online) gives one answer. In this case, I've created a stack object and filled it with the names of my cats. I then retrieve the size of the stack and print out it's contents using a simple for loop and the "document.write()" method. Of course, you don't have to put cat names in the stack. Thanks to JavaScript's loose data typing, your stacks can contain strings, numbers, other objects, or whatever else you might want. ■

Quantity reprints of this article can be purchased by phone: 717.560.2001, ext.39 or by email: sales@rmsreprints.com.

Immutables

Mark Davis
mark@macchiato.com

Dr. Mark Davis is a program director at IBM's Center for Java Technology, Silicon Valley. Mark co-founded the Unicode effort, and is the president of the Unicode Consortium. He is a principal co-author and editor of the Unicode Standard, Versions 1.0 and 2.0, and architected the bulk of the Java JDK1.1 internationalization support.

Recently, my family played a hilarious set of Dictionary games. The rules are simple. You need an unabridged dictionary, pencils, and paper. In each round, one person is the dealer. He/she looks up an obscure word in the dictionary and writes down its definition. The other players write down a counterfeit definition and pass to the dealer. The dealer reads off all definitions, and players vote for one (other than their own).

For each vote for a counterfeit definition, the dealer and the counterfeit's author each get a point. For each vote for the right definition, the voter gets one point. The strategy is for the dealer to pick a word with the most *implausible* real definition, and the other players to write the most *plausible* counterfeit definition. You get some really creative counterfeits—and some very funny real words.

Let's take an example: which of the following looks the most plausible?

im·mu·ta·ble /i(m)·'myû·tù·bùl/ adj. Not capable of or susceptible to change. [Middle English, from Latin *immutabilis*, from *in-* + mutabilis mutable. 15th Century.]

> That by two immutable things, in which it was impossible for God to lie, we might have a strong consolation. Heb. vi. 18.

im·mu·ta·ble /i(m)·'myû·tù·bùl/ adj. Not capable of being mute. [extension of Middle English *muet*, from Middle French, from Old French *mu*, from Latin *mutus*, probably from *mu*, representation of a muttered sound. 19th Century.]

> Percy blathered on and on despite Lizzy's best efforts to restrain him. "That man is immutable!" she thought. *Jane Austen*

For Java, the correct answer (the first definition) is very important, because the concept of immutable objects is crucial to designing durable APIs—and also a major factor in performance.

So what are immutable objects? They are objects that cannot be changed once they are created. There are a number of significant advantages to them:

- They behave more like primitive types.
- They are automatically thread-safe.
- If you are passed an immutable object, you never have to worry that someone else will change it behind your back.

The main disadvantage of immutable objects can be performance, so it is important to know when and where to use them.

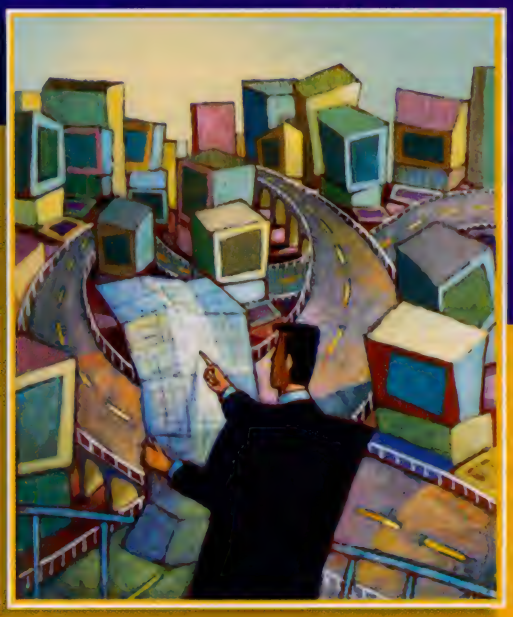
We don't have to look far for examples: the Java API provides two very similar classes, one immutable and one mutable. String was designed to be immutable, and thus simpler and safer. StringBuffer was designed to be mutable, with tremendous advantages in performance. For example, look at the following code, where one uses string concatenation and the other uses the StringBuffer append method. (The doSomething() method is overloaded to take characters as either Strings or StringBuffers.)

```
Case 1
for (i = 0; i < source.length; ++i) {
    doSomething(i + ": " + source[i]);
}

Case 2
StringBuffer temp = new StringBuffer();
for (i = 0; i < source.length; ++i) {
    temp.setLength(0); // clear previous contents
    temp.append(i).append(": ").append(source[i]);
    doSomething(temp);
}
```

Behind your back, the compiler will optimize Case 1, but it still involves the creation of three objects per iteration. Case 2, on the other hand, avoids any object creation. Because of this, Case

Hard Facts. Advanced Technologies. Step-by-Step Instruction.
All geared to increasing your productivity.



Introducing:

APPLICATION DEVELOPMENT'99

The European Event for Building Distributed Systems

Incorporating **Object Expo-Europe/SIGS Expo for Java**

Conference: **5-9 July 1999**
Exhibit: **7-8 July 1999**

Queen Elizabeth II Conference Centre, Westminster, London

The Natural Evolution of Object Expo-Europe

With the widespread acceptance and maturity of object technology, Object Expo-Europe has now evolved into Application Development '99. Its goal is to give you what you need to know to increase your overall software development productivity. To accomplish this, we've carefully designed a technical programme that provides you with the latest techniques and valuable advice for utilizing the most powerful technologies emerging. Celebrate our 10th year with us, and version 2.0 of Object Expo-Europe -now Application Development '99.

Objective, Technical and Platform Neutral

Select from an unprecedented 100+ sessions from the following tracks:

- CBD Solutions
- Advanced C++ Programming
- Distributed Object Strategies
- Mastering Patterns and Frameworks
- Windows Programming Tips and Tricks
- Effective Database Integration
- Java Programming Fundamentals
- Distributed Java Strategies and Tactics
- Software development Techniques
- Java for Gurus

Acclaims from Last Year's Delegates

"Excellent presentations - rich precise information."
Wolfgang Ponihuan,
System Designer
Daiuile-Benz Aerospace AG

"Provided clear, real-world analogies."
Diane Martini,
Project Manager
Parallax Solutions Ltd.

"Practical tips. Excellent presenters."
Edwin Lynzaad,
Consultant
Vitalogic BV

What you'll learn (Partial listing)

- Requirements Engineering
- Effectively using UML 1.1 and UML Process
- Enhancing your projects with Business Component Analysis
- Advanced use case analysis for software requirements
- Using agents
- Building large scale server apps with EJB
- Effectively using MTS and ATL
- Comparing Distributed Object Models- COM, CORBA, RMI
- Distributed object debugging
- Developing a component framework with CORBA
- Mastering patterns and frameworks
- Understanding COM
- Mapping objects to relational databases
- Using Java threads
- Implementing XML in Java
- Using components with Java
- Integrating Java into legacy systems
- Programming the Java layout manager
- O-O A&D in Java
- Building Java objects in CORBA
- Advanced server-side Java programming
- Getting the most out of EJB
- & much more.

For additional information, contact SIGS Ltd:

- **Tel:** +44 (0) 1306 631331
- **Fax:** +44 (0) 1306 631696
- **Email:** enquiries@sigs.com

Visit us online at:
www.applicationdevelopment.com

Corporate Sponsors:

PLATINUM TECHNOLOGY

COMPUTER ASSOCIATES

Premiere Media Sponsor:

computing

Presented by:

SIGS CONFERENCES

Media Sponsors:

Application Development Advisor

Component Strategies

Java Report

JOOP

UnixNT

IBM System User

Dr. Dobb's Journal

Durable APIs

Designing for larger-scale systems can be quite a bit different than for small programs. In particular, the cost of making incompatible changes becomes very high.

This means that you have to design your code to be durable—where the APIs can withstand the test of time, and can grow when you need to extend them, without breaking your clients.

2 can be more than 1000% faster than Case 1 (depending on your JIT, of course). So let's take a little time out to talk about object creation.

AVOIDING OBJECT CREATION

Many articles have been written about just how bad excessive object creation can be for your performance. Object creation alone will represent a significant proportion of the CPU time taken by your program. This time is so important that for most programs it is a valuable proxy for the overall speed—when doing performance analysis and enhancements, reducing object creation by 10% generally corresponds to a 10% increase in speed!

The lifetime cost of an allocated object includes a number of items you don't often consider. Think in terms of the *lifetime* cost of object creation, not just creation time alone:

- Cost to initially allocate the storage
- Cost to examine the object during each garbage collection cycle while it is alive
- Incremental extra virtual memory swapping and garbage collection required because memory is somewhat tighter
- Cost to recover the storage when its life is over

Even though the lifetime cost may not be noticeable when your program is run in isolation, it can still matter. Nowadays, there are more and more programs running concurrently on the same machine. If your program eats up extra CPU cycles, it can affect the overall performance of the user's machine. While garbage collection may operate during lulls in your threads' operation, it will still affect the performance of other programs running concurrently on your machine.

PASS-IN, PASS-OUT IDIOM

One of the goals of your API should be to not *require* unnecessary object creation. Let's take a look at a simple example from Component to see how to do this. (The implementation given here is simplified slightly.)

```
public Dimension getSize() {
    return new Dimension(width, height);
}
```

Because the method has a functional return, I can have a complex expression involving the return value from this function. For example, here we find the horizontal position of a piece of text which is centered in a component.

```
x = (myComponent.getSize().width -
        metrics.stringWidth(s))/2;
```

Other examples of *chaining* functional returns are:

- pane.getHAdjustable().setUnitIncrement(cellWidth);
- return Integer.toString(ch,16).toUpperCase();
- myMenu.getItem(i).setState(state);
- String fonts[] = Toolkit.getDefaultToolkit().getFontList();
- setFont(new Font(getFont().getName(), Font. BOLD, getFont.getSize());

If Dimension were an immutable class, then a functional return of an object in a getter would have no cost. (Presumably, it would just be one of the component's fields, which is simply returned.) By being immutable, this would be a safe operation—the caller couldn't corrupt the internals of the component.

However, Dimension is not immutable, so Component is forced to create a new object each time this method is called. In this particular case, the drain on performance is probably small enough to be acceptable—but often it isn't.

One option for not forcing object creation is to make the Dimension an output parameter instead:

```
public void getSize(Dimension output) {
    output.width = width;
    output.height = height;
}
```

This avoids the requirement of object creation—or rather, it puts the responsibility on the caller. So why is this more efficient? Because the caller can then reuse the same Dimension object for a whole slew of code.

However, there are a lot of circumstances where a functional return is very handy for the programmer. Even where we have previously allocated a temporary Dimension, many people dislike breaking up the code into multiple statements. Look at how an output parameter changes our previous example:

```
myComponent.getSize(temp);
y = (temp.width - metrics.stringWidth(s))/2;
```

One way to get around this is to use the *pass-in, pass-out idiom*. With this idiom, you return the output parameter from the routine. The implementation becomes:

```
public Dimension getSize(Dimension output) {
    output.width = width;
    output.height = height;
    return output;
}
```

You can then provide your original API as a convenience method. By making it final, any good JIT will compile out any overhead.

```
public final Dimension getSize() {
    return getSize(new Dimension());
}
```

Your clients then have the speed advantages of not forcing object creation and the convenience of functional returns. (The cost of having a return value is very small.) The example becomes the following—if someone is very concerned about performance (anyone not concerned with performance can continue to use the original API).

```
y = (myComponent.getSize(temp).width -
        metrics.stringWidth(s))/2;
```

☞When setters have functional returns, the normal return value is the main object itself, e.g., return this. For example, StringBufferappend(...) returns itself, which allows functional chaining such as:

```
result.append(", ").append(s);
```

Let's summarize some of the main points about object creation before we turn back to immutables.

- Avoid unnecessary object creation. Think of the lifetime cost of object creation, not just creation time alone.
- Use output parameters to avoid requiring the creation of mutable objects.
- Use the pass-in, pass-out idiom in that case, if functional returns are required.
- Provide simple convenience methods, returning objects where useful.

DEFINING IMMUTABLE OBJECTS

We saw that immutable objects are easier to return. So do they solve all our problems? Unfortunately not. The big disadvantage of immutables is: *to make one with a different value, you have to create a whole new object.*

For example, let's set two fields from different objects to the value 3.

```
myDimension.width = 3;
myColor = new Color(myColor.getRed(),
                    myColor.getGreen(), 3);
```

Dimension is mutable and can just be set directly. Color is not, and a whole new object must be created. For a feeling of what that represents, look at a Mandelbrot program drawing on the screen. Because essentially every point displayed is a different color, repainting a 200 x 200 window will create (and eventually destroy) 40,000 objects.

SAFE IMMUTABLES

The first step in making an object immutable is to make sure that there are no setters; only the constructor will change the values of any fields. Next, make all fields private. This is good advice for any class but is crucial for immutables so that subclasses and other classes in the same package can't muck with your fields. There is one exception to this rule: you might have a lazy-evaluated field, such as a cache, whose value changes after construction time. This is okay as long as (1) semantically the object always behaves the same after construction, and (2) all accesses to that field internally are synchronized.

Creating a safe, truly immutable object can be expensive in some circumstances. If any getter returns a mutable field, that return value must be cloned. Otherwise the object is *not* truly immutable. Look at the following example:

```
public class AClass {
        public int[] getItems() {
            return items;                    // error
        }
```

Get the latest Java™ tools delivered right to your computer.
This Week in Java™, your FREE weekly e-mail newsletter will keep you on the cutting edge of Java™ development.
• Special offers from your favorite vendors
• Product trials
• Insider news and tidbits
Sign up today!
www.javareport.com

```
    private int[] items;
}

...
int[] myItems = aClass.getItems();
myItems[1] = 0;                    // modifies aClass!
```

Dealing with arrays is particularly unpleasant because there is no good way to make an immutable version of an array. If you pass back an array—or any other mutable—from an Immutable class, you have to clone it.

```
return (int[])items.clone();  // corrected
```

Cloning setter results may be obvious to you. What may not be obvious is that when any mutable objects are passed into the constructor, they must *also* be cloned. Otherwise the object is *not* truly immutable. Take this example:

```
public class AClass {
    public AClass(int count, int[] items) {
        this.count = count;
        this.items = items;            // error
    }
    private int count;
    private int[] items;
}

...
int[] myItems = {1, 2, 3};
AClass aClass = new AClass(2, myItems);
myItems[1] = 0;                    // modifies aClass!
```

To fix this, the bad line has to be changed to:

```
this.items = (int[])items.clone(); // corrected
```

Cloning is not always an easy answer, either. Many mutable classes do not support a *safe clone*; even after cloning, changes to the original object can cause changes to the cloned object or vice versa. For example, Vector.clone() is not safe. You have to look at each class in question to make sure that the clone method is safe. If it is not safe, then you can either avoid using the class in construction or getters, or write your own safe clone (where possible). This hits on the topics of deep-cloning and object ownership, which I'll take up in the future.

☞Notice that partial immutability is like bad home insurance—*worse than nothing*! If a class purports to be immutable but is not, then you don't take the precautions that you would with a mutable object, producing nasty, intermittent bugs. You depend on it, but it fails you just when you need it.

DUAL CLASSES

If Java had the const keyword, as in C or C++, immutability would not be much of an issue. However, it doesn't, it probably will never, and there is no use whining about it. We just have to make do.

☞For those of you familiar with C or C++, an immutable class is actually much stronger than const. With a const parameter, for example, you can't change the parameter, but you have *no* idea whether someone else can behind your back. Moreover, const can always be cast away, making your code fragile.

Warning: *static final* is **not** the same as *const*. *Look at the following*!

```
public static final Point ORIGIN = new Point(0,0);
void someMethod() {
    ORIGIN.x = 3; // alters ORIGIN—no problem!
}
```

For the clients of your class, the best way to address the issue is for you to supply both kinds of objects (mutable and immutable) with a common base class. This can supply the advantages of both to your clients. For example:

```
public class ColorBase {
    public int getRed() {...
    ...
}
public class Color extends ColorBase {
    // current API, plus...
    public Color(ColorBase other) {...}
    ...
}
public class ColorBuffer extends ColorBase {...
    public int setRed() {...}
    ...
    set(ColorBase color) {...}
    ...
}
```

☞Important! Don't make the mistake of supplying a base class that purports to be immutable with a subclass that is not. For example, suppose that String was not final and had subclasses like StringBuffer.

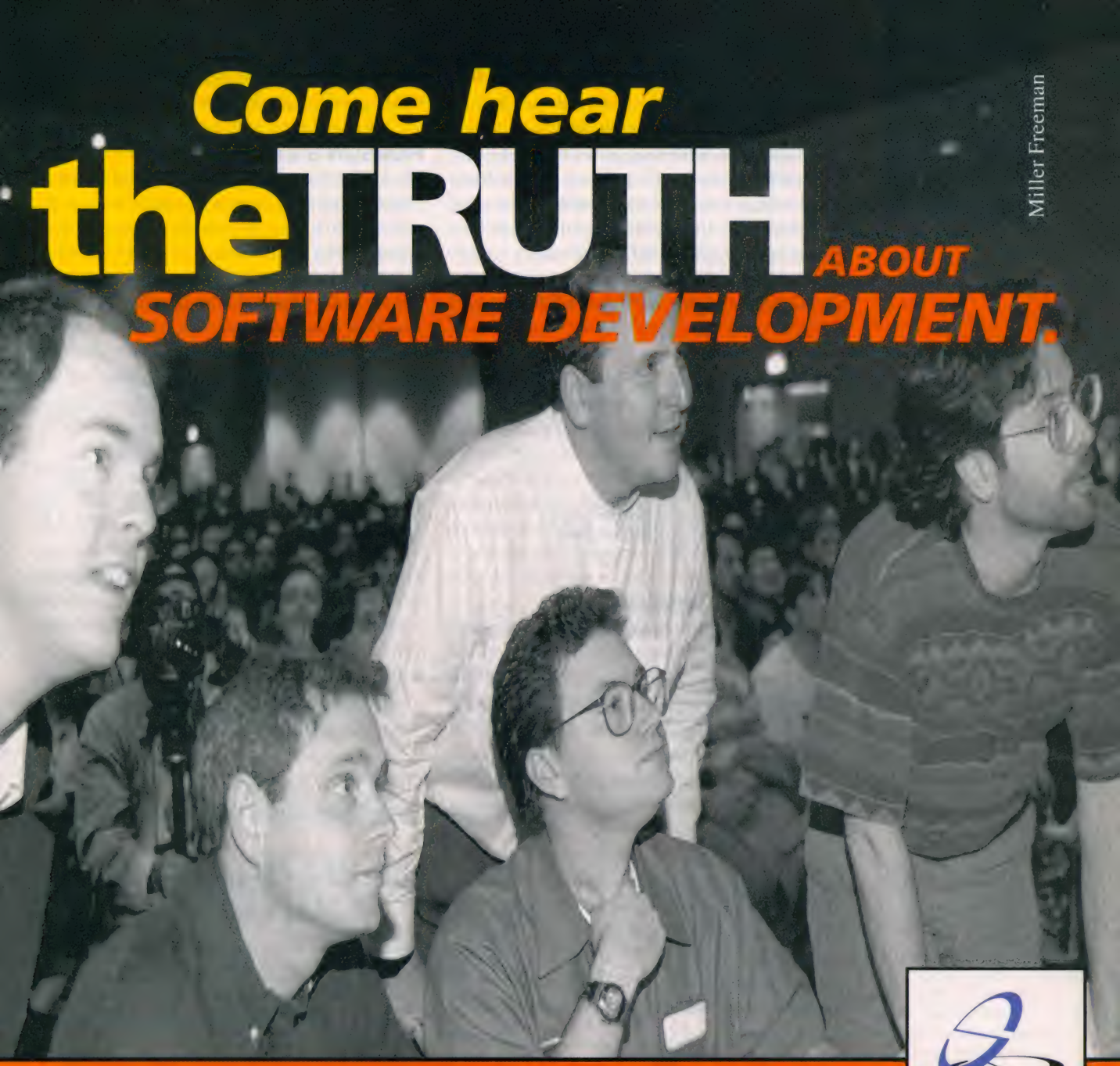
If I got passed an object of type String, I wouldn't know whether the actual object was immutable or some mutable subclass that someone else could change behind my back. I don't know whether I need to clone the object for thread-safety or not!

Providing this structure involves adding a constructor to Color that takes a ColorBase. The new class ColorBuffer should supply setters for all the fields that you are able to set with the constructor for Color. In addition, it should supply set(ColorBase color). It may optionally supply overloads of set that take some of the sets of constructor arguments for Color, such as:

```
set(int red, int green, int blue);
set(float red, float green, float blue);
```

With these three classes, clients could then return (or convert to) Color when they needed immutability, and use ColorBuffer if not. Methods

Come hear the TRUTH ABOUT SOFTWARE DEVELOPMENT.



Miller Freeman

THEN CHALLENGE THE INDUSTRY EXPERTS WHO CREATED IT.

The thing that makes Software Development '99 so exciting is the very same thing that makes it different from any other conference.

SD '99 is the only major forum for development professionals that is truly independent. Instead of a single point of view, you'll hear industry leaders representing a variety of technologies and a spectrum of competing visions. Then you'll evaluate and extract the truth for yourself.

SD '99 isn't a marketing event. It's about keeping developers ahead of the curve, well-rounded and more effective. Which is why, unlike any other, this conference has flourished for over a decade.

This year, choose from over 200 classes and tutorials in Java, C++, Internet development, Objects and Components, Modeling Techniques, Usability Issues, Management Practices and more. And test-drive the latest tools from over 250 vendors.

So join thousands of developers and their managers at the most critical event in the industry – the one that focuses on developing the developer.

SD|99 software DEVELOPMENT

SOFTWARE DEVELOPMENT '99

San Francisco
Moscone Center

Exhibition
May 10-12

Conference
May 9-13

GET THE BIG PICTURE.

call for registration information *(800) 441-8826* email us at *sd99@mfi.com* or visit *www.sdexpo.com*

in other classes that didn't care about immutability or mutability could take parameters of type ColorBase. This framework allows for using immutables where necessary for safety, mutables where necessary for speed, and avoiding unnecessary object creation and conversion. The Java class library itself could allow programs to be written with *much* better performance if this were done: String and StringBuffer are the key example. We'll go into this in more detail in the future.

☞I recommend the simple name "set" for setters corresponding to a constructor. The Java API names for these setters vary over the map. For example:

```
myPoint.setLocation(otherPoint);
myRectangle.setBounds(otherRectange);
myDimension.setSize(otherDimension);
```

MODIFYING IMMUTABLES

"What? You've been saying all along that you can't modify them!"

That's right. You don't really modify them. But often you want to produce the effect of modifying them by creating a new object based on them, with some modification. For example, to add a character to a string, I set the variable to point to a new string created by using the old one:

```
str = str + "a";
```

In this simple (and common) case, I don't care that the object formerly pointed to by the variable str has not changed. As far as I am concerned, str now has an "a" at the end, where it didn't before. Similarly, with BigDecimal I replace the value of a variable with a newly created one:

```
aBigDecimal = aBigDecimal.divide(otherBigDecimal);
```

We really need a new term for "producing the effect of modifying." Our trusty online *Webster's* gives us quite a range of choices for such a term:

> *alter*, modify, mutate, refashion, turn, vary;
> *convert*, metamorphose, transform, transmute; diversify, variegate;
> *transform*, commute, convert, metamorphose, transfigure, translate, transmogrify, transmute, transpose, transubstantiate;
> *sterilize*, castrate, desexualize, fix, geld, mutilate, neuter, unsex;
> *exchange*, replace, substitute, swap, switch, trade, interchange.

None of them really fit, though a few would be pretty interesting to toss around in design discussions! So for now, until some helpful reader supplies a better term, I'll use the phrase "produce a modification."

HOBSON'S CHOICE

If you don't want the extra overhead or hassle of providing dual classes, then you have to decide whether to make your class mutable or immutable. The trade-offs are not always clear-cut. The factors that will tend to push you towards or away from immutability depend on the common usage patterns.

Pros: factors for immutability

- Modifications of your object are not commonly produced.
- Your object will often be used in getters and setters on other objects that have to be safe. Remember that immutables don't need to be cloned in getters and setters.
- Your object doesn't tend to have lots of fields. Producing modifications with lots of fields means some complicated constructors, which get unwieldy.
- The fields of your object are all immutable or primitive. If not, you will have to clone in getters and setters, costing some performance in those operations.
- Best if your class is final. That way clients won't have to make runtime type checks.

Cons: factors against immutability

- Modifications of your object are commonly produced.
- Your object won't often be used in getters and setters on other objects, or those objects won't care if their fields change contents without their knowledge.
- People need to retain references to your object across changes to it. For example, making TextArea immutable wouldn't do; any change to it would mean replacing it with another class and updating the components that refer to it.

NO SUPPORT

Unfortunately, there is one other complication. The Java language and class libraries have no direct support for immutability. Surprisingly—for a concept that is crucial to effective use of the Java class libraries—classes are immutable only by convention. This means that as a class designer, you must depend on documentation to show that your class is immutable; you cannot mark your class as implementing Immutable, as you can with Cloneable, for example.

One way to address this, at least within your

own classes, is to introduce your own Immutable interface, allowing users of your classes to test for immutability at runtime with instanceof.

Unless you document clearly or use the interface mechanism, when you have a class that looks like it is immutable, your subclassers may depend on it being so. Yet this might only be a coincidence, because you didn't supply setters on the first release. But once you have done that, you are stuck. Your clients may be depending on immutable behavior from your object, so you shouldn't make any changes in the future that make it mutable because it might break their subclasses.

On the other hand, if you are a subclasser, drive defensively. Try not to depend on a class being immutable unless it is clearly documented that it is so. If you *have* to make an assumption in the absence of documentation, ensure that there are no setters (any methods that can change state), and check that all fields are private (and not exposed through getters): no public, protected, or package-private fields. If the class is not final, remember that when you get an object of that type, it might actually be a subclass that *is* mutable. ■

Quantity reprints of this article can be purchased by phone: 717.560.2001, ext.39 or by email: sales@rmsreprints.com.

POWER JAVA *continued from page 65*

Unfortunately, this is extremely difficult to do properly. In your method, you may have called many other methods that have changed many states of objects. If you throw an exception and leave your method, you will have quite a bit of work to clean up to get back to a state where your code is valid again. This problem is not easily solved and requires much thought and effort. It is the single biggest inhibitor in producing really robust Java code that utilizes exceptions properly.

SUMMARY

Exception handling, although useful, is not a cure-all for your error handling problems. There are many areas of exception handling that are not very obvious and can lead to more complexity and more problems. Used properly and carefully, exception handling has many benefits. It has its share of pitfalls, and these can and should be avoided through judicious use of these concepts. ■

Peter Haggar is an Advisory Software Engineer with IBM in Research Triangle Park, NC. He can be contacted at haggar@us.ibm.com.

Quantity reprints of this article can be purchased by phone: 717.560.2001, ext.39 or by email: sales@rmsreprints.com.

Java CORBA

Alta Software, Inc. is a premier consulting firm specializing in e-commerce solutions and distributed object-oriented business applications for its clients nationwide. Because we are staffed by highly-talented technical professionals with a wealth of experience in the OO world, Alta's reputation in the industry is stellar.

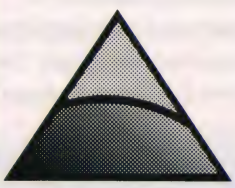
We are seeking talented individuals with experience designing, developing, and delivering technical solutions using Java, C++, CORBA, OO/DBMSs, and similar OO technologies. Positions available in northern California, Chicago, Atlanta, and Northern Virginia.

Alta offers its employees not only highly competitive compensation and bonus packages, but an ownership stake in the company and exceptional career development opportunities as well. For more information, please send your resume to:

hr@altasoft.com
Alta Software, Inc.
11480 Sunset Hills Road
Suite 200E
Reston, VA 22090
FAX: (703) 742-0751

EOE

Visit our Web site at:
www.altasoft.com



Alta Software, Inc.

Be sure to mention that you saw this ad in Java Report

PASSIONATE ABOUT JAVA?

Do you want to share it with others?

If so, DevelopMentor wants you to join our team of expert instructors. You don't have to leave your current employer, but you do have to love the technology, have a true mastery of it's concepts and details, a desire to share your knowledge with others, and time to deliver at least one 5-day course a month.

DevelopMentor specializes in setting the standard for instructor-led training courses. DevelopMentor courses are offered at our training facilities in Southern California, Boston, Portland and the UK, and at customer sites worldwide. Complete curriculum information is available on our web site. **www.develop.com**

For more information about available positions contact Peg Bennett at pegb@develop.com.

DEVELOPMENTOR™

SCOTT'S SOLUTIONS *cont'd. from page 80*

```
            BufferedReader br =
                    new BufferedReader(fr);
            String className =
                        br.readLine();
            Class cl =
                Class.forName(className);
            Calc c = (Calc)
                    cl.newInstance();
            fr.close();
            return c;
        } catch (Exception e) {
            return null;
        }
    }
```

Changing implementations of classes is now simply a matter of writing a new class (e.g., class CalcImplV1 implements Calc) and placing the name of the class (CalcImplV1) into the correct file.

We must take care in this example that at an operating-system level, changing of the file is an atomic operation. If the file were partially written, then the data in the file could be corrupt and the appropriate class would not be found. With a simple file like this, that's rarely a problem. Alternately, however, we could hold the name of the class in a database; this would completely ensure that changing the name of the class would be an atomic operation.

While this is a completely usable technique, we have cheated a little bit. The end result of this operation is that we might have two clients that are simultaneously connected to our server; one client might be executing the code contained in CalcImplV1 while another is executing the code contained in CalcImplV2. From the point of view of the client, we've achieved our goal: each client is executing different code simultaneously.

From the point of view of the server, however, we just have two distinct classes, each of which implements a common interface. So we don't really have two implementations of the same class that are active simultaneously in our server; we've just approximated that goal. And for many purposes, that's good enough.

However, there's another route that we can take, one that offers a little more flexibility. This route depends on a little-known feature of the VM, which is a property of the class loader that is used to create Java classes. The VM makes a distinction between classes that are loaded by different class loaders; we'll use this feature to obtain the versioning control we're after.

The feature of the class loader may be little known, but it is rather intuitive. Let's say that you surf to www.acme.com, and their home page runs an animation applet, the class name of which is Animate. Then you surf to www.xyz.com, and their home page contains a different animation applet, the class name of which also happens to be Animate.

Clearly, these are two different classes, even though they happen to have the same name. Something must be happening inside of the VM to differentiate between these two classes, and that something is an artifact of their having been loaded by different class loaders. Inside the VM, a class is defined by its class name, its package name, and the instance of the class loader that was used to find it.

We can use this property of the class loader to rewrite our getCalc() method like this:

```
public static Calc getCalc() {

    try {
        URL urls[] = new URL[1];
        urls[0] = new URL(...);
        Class c = CalcServer.class;
        ClassLoader cl =
                c.getClassLoader();
        URLClassLoader ucl =
                new URLClassLoader(
                    urls, cl);
        c =
            ucl.loadClass("CalcImpl");
        Calc calc = (Calc)
                    c.newInstance();
        return calc;
    } catch (Exception e) {
        return null;
    }
}
```

Assuming that we provide the correct location for the URL from which to load the classes, now every client will load its classes from a new instance of the URLClassLoader class. Hence, any classes that are loaded by one client connection will be distinct from any classes that are loaded by another client connection, even if those classes have the same name.

So now, to substitute a new implementation of our calculation server, we simply have to recompile any classes that have changed. We're

Java™ Report (ISSN 1086-4660) is published monthly by SIGS Publications, 71 West 23rd St., 3rd Floor, New York, NY 10010, 212.242.7447. © Copyright 1999 by SIGS Publications, Inc. All rights reserved. Periodicals postage paid at New York, NY, and additional mailing offices. Regular subscription rates— $39 domestic, $64 Canada and Mexico, $88 foreign; institutional—$139 domestic, $164 Canada and Mexico, $188 foreign. Postmaster: Send address changes and subscription orders to: Java Report, P.O. Box 5050, Brentwood, TN 37024-5050. GST (Canada) #R895884229. For service on current subscriptions: in North America call 800.361.1279 (v) or 615.370.4845 (f); (custserv@cwixmail.com (email); outside North America +44(0)1858 435 302 (v), +44(0) 1858 434 958 (f); custserv@cwixmail.com (email). Statements of opinion and fact are made on the responsibility of the authors alone and do not imply an opinion on the part of SIGS Publications, Inc. or the editorial staff. All trademarks are the property of their respective owners. Printed in the USA.

Java and Java-based marks are trademarks or registered trademarks of Sun Microsystems Inc. in the United States and other countries. *Java™ Report* is independent of Sun Microsystems, Inc.

Please send unsolicited manuscripts to Managing Editor, *Java™ Report*, 71 West 23rd St., New York, NY 10010, email: kmajor@sigs.com.

not able to change the Calc interface, nor the server classes themselves, but the CalcImpl class, plus any classes that the CalcImpl class uses, will be distinct for each client connection.

In terms of the implementation of the target class—that is, the class that implements the Calc interface—there is little difference between the two techniques that we've just looked at. In either case, we can change the implementation of the class and new clients will use the new implementation.

The advantage of the second technique, however, is that it is much easier to change any classes that the target class depends on. Say for example that our implementation of the Calc interface looked like this:

```
public class CalcImpl
                    extends Calc {
    Socket s;
    public void setSocket(Socket s){
        this.s = s;
    }
    public void run() {

        Matrix m = new Matrix();
        m.populate(s);
        ...
    }
}
```

If we're interested in changing the implementation of the Matrix class, our first technique won't work: the Matrix class in that case has already been defined. What we'd need to do for our first technique is to create an interface for the Matrix class and have the CalcImpl class use another factory interface to obtain instances of the Matrix class. And if the Matrix class uses Row and Column classes that might change, they too would need to be defined by interfaces and obtained via factories.

It's certainly possible to write an application like this, but it's usually inconvenient—what if there were 50 such classes in our application? By using the class-loader technique, we have much more flexibility: we can simply overwrite any method or methods for which we want to provide a new implementation. Because all the new implementations will be loaded by a new instance of the class's loader, they will not interfere with the previously-loaded implementations.

The tricky part about using the class-loader technique is to ensure that all the classes we're interested in are changed atomically. We had that concern when all we were changing was a single line in a file, but we could generally ignore it because operating systems write files in terms of blocks, and the block would change atomically from the point of view of the server.

When you're changing an entire file—or multiple files—that's not necessarily the case. In the worst case, this may mean that you'll need to quiesce the server while changes are made to the file server (when the classes are loaded through a file-based URL) or shut down the Web server while changes are made to that Web server (when the classes are loaded through an HTTP-based URL).

Alternately, you can define a different class loader. We used the URLClassLoader in our above example because it was convenient to do so: it's a fully implemented class loader. But the URLClassLoader is only available in Java 2, and it can only load classes through URLs. If you wanted to load classes from a database (so that the set of classes could be changed atomically), or if you wanted to use the class-loading technique in Java 1, you'd have to implement your own class loader.

That's not a big obstacle—it's simply a matter of obtaining an array of bytes (the class file) and calling the defineClass() method inside your class loader. But the necessity of that is not absolute—the URLClassLoader is usually good enough for these purposes.

As an aside, we should also mention that the first technique has the advantage that it can be used by an applet. That's a somewhat dubious advantage; we're really interested in this topic in a long-running program that we can't afford to restart. But because applets cannot create class loaders (at least, not by default), they could not use the second technique even if they needed to.

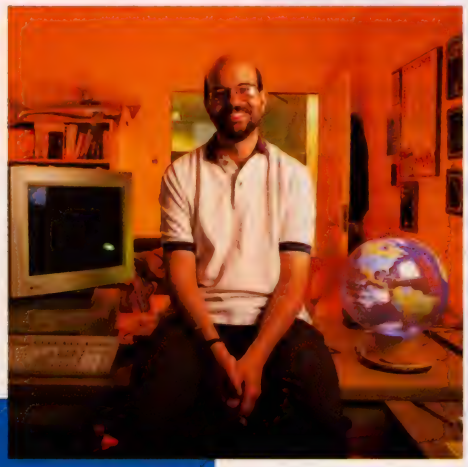
One other time when the class-loading option becomes important is in certain types of RMI servers. Because RMI servers have the ability to load class definitions from their clients, it is important that such servers use the class-loading technique that we've just explored to do so. Because different clients may ask the server to download different classes with the same name (much like different pages in a Web browser referenced different classes with the same name), an RMI server that downloads classes must do so in a different class loader for each client.

Fortunately, there is an RMIClassLoader class that can be used for this purpose; when you call the static loadClass() method of that class, a new class-loader will be created automatically. So the RMIClassLoader class does the right thing transparently, but if you're really interested in this topic, you can use its implementation as another guideline for how to attack the problem of different versions of the same class simultaneously. ■

Quantity reprints of this article can be purchased by phone: 717.560.2001, ext.39 or by email: sales@rmsreprints.com.

Versioning solutions

This column is designed to give readers solutions to questions with a "how-to" bent. If you'd like to submit a question, send it to ***soaks@sigs.com*** *or visit* ***http://www.sigs.com/java/ask.html****. In each issue, we'll answer the most common questions about Java programming.*



Scott Oaks
soaks@sigs.com

THIS TIME, WE examine two approaches that can help solve the versioning problem. That's actually somewhat inaccurate—versioning issues creep into several areas of Java. When objects are deserialized, there is a potential versioning problem if the implementation of the serialized class has changed; there are version numbers in the Java class file specification that can come into play when you deal with low-level issues of the virtual machine (VM) and there are probably others as well.

We're not interested in those issues (at least not today). The versioning problem that we're interested in addressing occurs with a long-running server. There are times when a server is running that you want to change the implementation of one of the classes that it is using without disturbing the original class. Think of a server handling requests from several clients: at one point in time, we'll discover that we need to fix a bug or for some other reason change one of the classes that the server is using to process a client request.

We could do that by terminating the server, but that would disrupt the current clients. We could wait for the current clients to finish the requests, then terminate and restart the server, but that leaves a gap in time when our server cannot perform its function. What we want to do is have the current clients continue with the original implementations and have any new clients use the new implementations. Ideally, there's a point in time at which we'll have two implementations of the same class within the server at the same time.

With a little bit of planning, we can do just that. One way to do it is to define the interaction between the client and server using an interface and having each client connection obtain its interface through a factory. Then we can dynamically change the class that the factory provides to the client.

Let's look at an example. The first thing that we must do is define the interface that objects of our factory will create:

```
public interface Calc
            extends Runnable {
    public void setSocket(Socket s);
}
```

The interface can be as simple or as complex as necessary; in this case, we're taking advantage of the fact that the Runnable interface already provides a convenient entry point—the run() method—for our calculation, but we typically need some way to set up initial parameters. Because its an interface, these parameters cannot be set through a constructor, so we must set them through other methods—the setSocket() method in this case.

Now we can implement the main() method of our server:

```
public static void main(
                    String args[]) {
    try {
        ServerSocket ss =
                new ServerSocket(8989);
        while (true) {
            Socket s = ss.accept();
            Calc c = getCalc();
            c.setSocket(s);
            new Thread(c).start();
        }
    } catch (Exception e) {}
}
```

Like many servers, our calculation server simply sets up a new thread for each client and runs that client in a new thread. The key is the fact that code for each client comes from the getCalc() method. This allows us to provide a different class for every client that connects to the server.

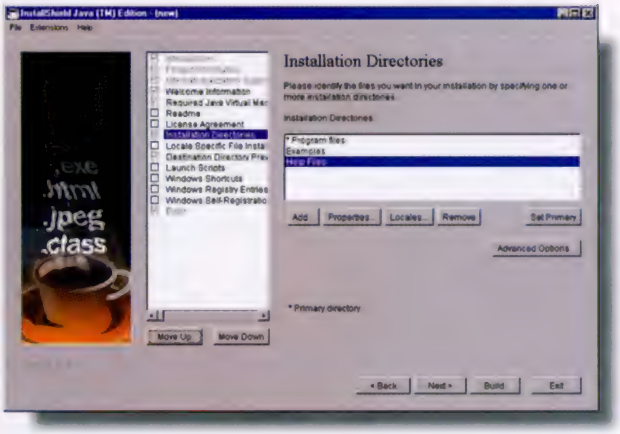
One way to achieve this is to keep in a file the name of the class to be instantiated. Then we could implement the getCalc() method like this:

```
public static Calc getCalc() {
    try {
        FileReader fr =
        new FileReader("classImpl");
```

continued on page 78

Scott Oaks is a Systems Engineer for Sun Microsystems, where he focuses on practical applications of Java technology. He is the co-author, with Henry Wong, of *Java Threads*.

InstallShield® Java™ Edition Version 2.5

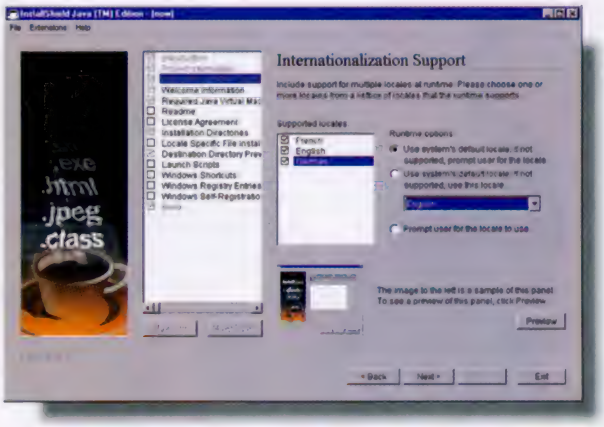


Multiple source and target directory support.

Through the InstallShield Java Builder, you can deploy files to multiple directories on a target system. As you build your setup, locate files in a clear, customized set of directories that will remain correctly distributed on your customer's machine after installation. No more cluttered directories!

Internationalization support.

You can display your custom extensions in different languages. InstallShield Java Edition now includes language resources for all standard end-user dialogs in French, German, Italian, Japanese, and Spanish. Use the **new internationalization APIs** to update your custom extensions, and your setup is ready for the world!



NEW VERSION!

InstallShield® Software Corporation

The total solution for Java™ deployment.

InstallShield® Java Edition provides developers with a visual tool designed to create a single Java installation that's consistent in look, feel, and functionality across multiple platforms. New InstallShield Extensions for the Java platform bring the power of Java to installation building, allowing you to plug your own Java-based extensions into your installation. And the InstallShield Installation Applet lets you create a seamless cross-platform launch site for your installation. Best of all, InstallShield Java Edition is authored by InstallShield Software Corporation, the worldwide leader in installation and distribution technology.

Exclusive features in InstallShield Java Edition 2.5

- True cross-platform installations—no need for multiple installations for individual platforms
- InstallShield Extensions for the Java platform give you the power of Java to build your installations—plug in your own Java-based extensions or use one of the pre-built examples
- InstallShield Installation Applet lets your users specify or get the appropriate Virtual Machine (VM), download, and install your Java application by clicking a Web link
- Self-extracting installations allow users to launch installation with point-and-click ease
- Command-line access allows you to add installations to your batch file building process
- Specify multiple source and multiple target directories for your installations
- Create installations in English, French, German, Italian, Japanese, and Spanish

©1999 InstallShield Software Corporation. All rights reserved. InstallShield is a registered trademark and service mark of InstallShield Corporation. "It All Starts Here" is a service mark and a registered trademark of InstallShield Corporation. Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and/or other countries. Part No. 181-20009 0199

It All Starts Here®

900 National Parkway, Suite 125
Schaumburg, IL 60173-5108 USA
Sales: +1-847-240-0618
Fax: +1-847-619-0788
Toll-free: +1-800-440-8149
sales@installshield.com
www.installshield.com

InstallShield® SOFTWARE CORPORATION

www.installshield.com/java

It won't mean a thing If it ain't got that swing

JCLASS JavaBeans

You need ***powerful components*** for professional Java development. But if they don't work with Swing and the latest JDK 1.2 release... they won't mean a thing.

JClass, the world's most popular suite of 100% Pure JavaBeans, now includes ***JClass SwingSuite***, the essential set of extensions and enhancements for Swing in JDK 1.2. And with the power of JClass Suites, you'll be building for the future ***today.***

JCLASS WORKS WITH:		
✓ JDK 1.0.2 (AWT)	✓ IBM VisualAge for Java	✓ Progress Apptivity
✓ JDK 1.1 (AWT or Swing)	✓ IST Visaj	✓ SilverStream
✓ JDK 1.2 (Swing)	✓ Microsoft Visual J++	✓ Sun Java WorkShop
✓ Bluestone Sapphire/Web	✓ NetBeans	✓ SuperCede
✓ Borland JBuilder	✓ Powersoft PowerJ	✓ Symantec Visual Café

New **JClass SwingSuite** - easily add the power of MDI, wizards and a robust collection of thread-friendly, lightweight components including JCTreeExplorer, JCTreeTable, JCSpinBoxes, a configurable DateChooser control and more!

- **JClass LiveTable** - the high performance Java grid/table for static, dynamic or db-bound data
- **JClass Chart** - the ultimate in Java charting and graphing, designed for superior performance
- **JClass HiGrid** - the innovative RAD outline-grid for hierarchical dynamic data
- **JClass DataSource** - a robust, hierarchical, multiple-platform data source
- **JClass Field** - complete input and validation for popular data types including calendar

Need rock-solid, robust Java GUIs? Professional Java developers around the world have made the award-winning JClass family of components their choice. Where else will you find seamless IDE integration and support for all major JDK versions? *Try JClass today for great performance with an improved swing!*

EVALUATE/ORDER ONLINE www.klgroup.com

KL GROUP
Software Development Productivity

KL Group Inc. Toll-Free: 1-888-361-3264 Tel: 416-594-1026 Fax: 416-594-1919
KL Group Europe B.V. Tel: +31 (0)20 679 95 03 Fax: +31 (0)20 470 03 26
www.klgroup.com

* Databinding feature requires purchase of JClass DataSource, or use of Borland JBuilder Professional or C/S edition, or Symantec Café dbDE edition. JClass, JClass Chart, JClass DataSource, JClass Field, JClass HiGrid, JClass LiveTable and JClass SwingSuite are trademarks of KL Group Inc. All other product names are trademarks or registered trademarks of their respective owners. JR004

FOR FASTER SERVICE & MORE INFORMATION:

FAX
615.370.4845

E-MAIL
subscriptions@sigs.com

ON-LINE
www.javareport.com

CUSTOMER SERVICE
1.800.361.1279

SIGS PUBLICATIONS

JR-APR-D

YES!

Enter my subscription at the term below. This is a risk-free offer. If I'm not completely satisfied I may cancel and get a prompt refund for the unmailed issues.

❑ 1 year—$39 (12 issues)

❑ **2 years—$62 (24 issues) a 20% savings**

○ **Check enclosed** (payable to ***Java™ Report)***

Charge my ○ Visa ○ Mastercard ○ American Express ○ **Bill me**

Card # Exp. date

Signature

Name

Title

Company

Address

City

State/Province Zip/Postal Code Country

Phone Fax

E-mail ○ Check here to receive only SIGS' promotions via e-mail

We reserve the right to rent our e-mail list

Make checks payable to ***Java™ Report***. Canadian and Mexican orders must add $25 per year. Checks must be in US dollars and drawn on a US bank. Canadian GST included (#R895884229)

Java and Java-based trademarks are registered trademarks of Sun Microsystems, Inc. in the United States and other countries. Java™ Report is independent of Sun Microsystems, Inc.

Java™ REPORT
THE INDEPENDENT SOURCE FOR JAVA DEVELOPMENT

SUBSCRIPTION SAVINGS COUPON

B90403

PLACE
POSTAGE
HERE

Java™ REPORT

THE INDEPENDENT SOURCE FOR JAVA DEVELOPMENT

PO Box 5050

Brentwood, TN 37024-5050

FOR FASTER SERVICE & MORE INFORMATION:

FAX

615.370.4845

E-MAIL

subscriptions@sigs.com

ON-LINE

www.javareport.com

CUSTOMER SERVICE

1.800.361.1279

SIGS PUBLICATIONS

For Faster Service & More Information:

FAX
615.370.4845

E-MAIL
subscriptions@sigs.com

ON-LINE
www.sigs.com/publications/cppr/

CUSTOMER SERVICE
1.800.361.1279

SIGS PUBLICATIONS

C++ REPORT SUBSCRIPTION SAVINGS COUPON

YES!

Enter my subscription at the term below. This is a risk-free offer. If I'm not completely satisfied I may cancel and get a prompt refund for the unmailed issues.

❑ 1 year—$79 (10 issues)

❑ **2 years—$126** (20 issues) a 20% discount

○ **Check enclosed** (payable to *C++ Report*)

Charge my ○ Visa ○ Mastercard ○ American Express ○ **Bill me**

Card # Exp. date

Signature

Name

Title

Company

Address

City

State/Province Zip/Postal Code Country

Phone Fax

E-mail ○ Check here to receive only SIGS' promotions via e-mail

We reserve the right to rent our e-mail list

Make checks payable to *C++ Report*. Canadian and Mexican orders must add $25 per year.
Checks must be in US dollars and drawn on a US bank. Canadian GST included (#R895884229)

B90403

Place
Postage
Here

C++REPORT

PO Box 5050
Brentwood, TN
37024-5050

For Faster Service & More Information:

FAX
615.370.4845

E-MAIL
subscriptions@sigs.com

ON-LINE
www.sigs.com/publications/cppr/

CUSTOMER SERVICE
1.800.361.1279

SIGS PUBLICATIONS

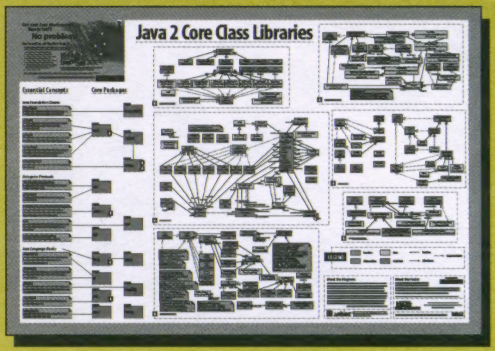
Unfold The Secrets of the Java 2 Class Hierarchy

FREE Poster

Get the **FREE** all new, illustrated, colorful, really great 38"x26" poster that details the core concepts of the Java™ 2 platform. From

SYMANTEC.™

www.javareport.com/poster



Name _____

Company _____

Address _____

City _____ State _____ Zip _____

e-mail _____ Phone _____

Quantities are limited so ACT now! Allow 3 weeks for processing.

We occasionally send e-mail messages with special offers from qualified, relevant businesses and organizations. Would you like to receive these messages?

❑ Yes ❑ No

SIGS' Conferences has a long tradition of presenting the world's most comprehensive, vendor-neutral, educational conferences focusing on advance software technology. Would you like to receive these messages?

❑ Yes ❑ No

PLACE
POSTAGE
HERE

Symantec Poster
Java™ Report
71 West 23rd St., 3rd floor
New York, NY 10010